An Adaptive Cell-Centered Projection Method
for the Incompressible Euler Equations

by

Daniel Francis Martin

B.S.M.E. (University of Florida) 1991

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering–Mechanical Engineering

in the

GRADUATE DIVISION

of the

UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Phillip Colella, Chair
Professor Van Carey
Professor James Demmel

Fall, 1998

The dissertation of Daniel Francis Martin is approved:

Phillip Colella, Chair	Date
Van Carey	Date
James Demmel	Date

University of California at Berkeley

Fall, 1998

An Adaptive Cell-Centered Projection Method for the Incompressible Euler Equations

© copyright 1998

by

Daniel F. Martin

(Blank Page)

Abstract

An Adaptive Cell-Centered Projection Method for the Incompressible Euler Equations

by

Daniel Francis Martin

Doctor of Philosophy in Engineering-Mechanical Engineering

University of California at Berkeley

Professor Phillip Colella, Chair

Adaptive methods for the numerical solution of partial differential equations concentrate computational effort where it is most needed. Such methods have proved useful for overcoming limitations in computational resources and improving the resolution of numerical solutions to a wide range of problems. By locally refining the computational mesh where needed to improve the accuracy of the solution, we more efficiently use computational resources, enabling better solution resolution than is possible with traditional single-grid approaches, representing a more efficient use of computational resources.

In this work, we present an adaptive cell-centered projection method for the incompressible Euler equations. It is an extension of the adaptive mesh refinement (AMR) methodology developed by Berger and Oliger for hyperbolic problems. Our algorithm is fully adaptive in time and space through the use of subcycling, in which finer grids are advanced at a smaller timestep than coarser ones. When coarse and fine grids reach the same time, they are then synchronized to ensure that the global solution is conservative and satisfies the divergence constraint across all levels of refinement.

Our method introduces three main innovations. First, we extend a cell-centered approximate projection discretization to a multilevel hierarchy of refined grids. Employing a cell-centered projection discretization permits the use of only one set of (cell-centered) solvers, which simplifies implementation and extension of this algorithm. Also, we use a volume-discrepancy scheme to approximately correct for advection errors due to the presence of coarse-fine interfaces. Finally, we synchronize coarse and fine levels by performing multilevel solves over all grids which have reached the same time.

Results are presented which show that the method presented in this work is second-order accurate, does not introduce instabilities due to the presence of coarse-fine interfaces, and which demonstrate the increased solution accuracy due to local refinement.

Phillip Colella, Chair Date

Contents

List of Figures vi

List of Tables x

Chapter 1 Introduction 1

1.1 AMR for Incompressible Flows 6
  1.1.1 Discretization Issues 7
1.2 Thesis Overview 11

Chapter 2 Finite Difference Methods on a Single Grid 14

2.1 Finite-Difference Notation 14
  2.1.1 Ghost Cell Implementation of Physical Boundary Conditions 16
2.2 Poisson's Equation 18
  2.2.1 Truncation Error Analysis 19
  2.2.2 Point Relaxation 22
  2.2.3 Multigrid Acceleration 27
2.3 The Incompressible Euler Equations 31
2.4 Projection Methods 32
2.5 Discretizing the Hodge-Helmholtz Projection 35
  2.5.1 The Discrete Projection 35
  2.5.2 Cell-centered Discretization of the Projection 37
  2.5.3 Different Projection Formulations 40
2.6 Single-Grid Algorithm 42
  2.6.1 Computing Advection Velocities 44
  2.6.2 Scalar Advection 46
  2.6.3 Velocity Predictor 47
  2.6.4 Projection 49
2.7 Filters 49
2.8 Convergence of the Algorithm 51

Chapter 3 Adaptive Solutions to Poisson's Equation 56
3.1 AMR Notation 56
  3.1.1 Proper Grid Generation 60
  3.1.2 Composite Operators and Level Operators 63
3.2 Solving Poisson's Equation on a Multilevel Hierarchy 71
  3.2.1 Composite Laplacian – Elliptic Matching 71
  3.2.2 Truncation Error Analysis 75
  3.2.3 Multilevel Multigrid Iteration Algorithm 77
  3.2.4 Extension to $n_{ref} = 2^p, p > 1$ 83
  3.2.5 Extension to $\ell_{base} > 0$ 84
  3.2.6 Level Solves 87
  3.2.7 Performance of the Algorithm 88
3.3 Alternate Algorithm 93
  3.3.1 LevelSolve + Correction Formulation 94
  3.3.2 Bottom-Up Iteration 95
  3.3.3 Top-Down Iteration 96
3.4 Convergence and Errors 104
  3.4.1 Convergence 104
  3.4.2 Effects of Local Refinement 107

Chapter 4 Adaptive Projection Algorithm 110
4.1 AMR for Hyperbolic Conservation Laws 110
  4.1.1 Conservation Laws 110
  4.1.2 Adaptive Methodology 112
  4.1.3 Recursive Timestepping Algorithm 119
4.2 Multilevel Discretization of the Incompressible Euler equations 121
  4.2.1 Level Algorithm 122
  4.2.2 Level Operators 123
4.3 A Simple Recursive Timestep 126
4.4 Additions to Hyperbolic Algorithm for Incompressible Flow 130
  4.4.1 Composite Projection 130
  4.4.2 Freestream Preservation 139
4.5 Complete Multilevel Algorithm 142
  4.5.1 Computing Advection Velocities 144
  4.5.2 Scalar Advection 145
  4.5.3 Velocity Predictor 146
  4.5.4 Level Projection 148
  4.5.5 Subcycled Advance of Finer Levels 148
  4.5.6 Synchronization 148
4.6 Initialization 151
  4.6.1 Comparison to Previous Work 156
4.7 Filters 159

Chapter 5 Error Estimation **160**
5.1 User-Defined Grids . . . 161
5.2 User-Defined Criteria . . . 161
5.3 Richardson Extrapolation . . . 163
5.3.1 Richardson Extrapolation for the Poisson Problem . . . 165
5.3.2 Richardson Extrapolation for the Time-Dependent Problem . . . 166
5.4 Grid Generation . . . 169
5.4.1 Clustering Algorithm . . . 170

Chapter 6 Results **172**
6.1 Test Problem Descriptions . . . 172
6.1.1 Single Vortex in a Box . . . 173
6.1.2 Traveling Vortex Pair . . . 173
6.1.3 Three Co-Rotating Vortices . . . 174
6.2 Passage Through Coarse-Fine Interfaces . . . 175
6.3 Volume-Discrepancy Correction . . . 176
6.3.1 Single Vortex . . . 176
6.3.2 Traveling Vortex Case . . . 181
6.4 Accuracy of AMR Calculations . . . 185
6.5 Performance . . . 191

Chapter 7 Software Implementation **197**
7.1 BoxLib . . . 198
7.2 Managing the AMR hierarchy . . . 198
7.2.1 Elliptic Solver . . . 200
7.2.2 Euler Equation Code . . . 201
7.3 Visualization . . . 203

Chapter 8 Conclusions **204**
8.1 Summary . . . 204
8.2 Conclusions and Future Work . . . 206

Bibliography **209**

List of Figures

1.1 Local refinement strategies: (a) cell-by-cell refinement, (b) block-structured refinement 4
1.2 Block-structured local refinement. Note that refinement is by a discrete amount and is organized into logically-rectangular patches. 7

2.1 Basic Cartesian finite-difference grid. Note that $\Delta x \neq \Delta y$ in this case. 15
2.2 Centering of finite-difference quantities. ϕ is cell-centered, η is node-centered, and F^x and F^y are edge-centered 16
2.3 Ghost cells outside computational domain. Solid cells are within the computational domain, while dashed cells are ghost cells outside the computational domain used to enforce boundary conditions. 17
2.4 Residual pattern which causes stalled convergence with point-Jacobi iteration 26
2.5 "Red-Black" ordering of cells for GSRB iteration 27
2.6 Pseudocode for a multigrid V-cycle 29
2.7 Max(Residual) vs. work units for a test problem 30
2.8 Edge-centered velocity field 35
2.9 Decoupled grids created by the exact DG operator 38
2.10 Left and right extrapolated states 44
2.11 Non-physical velocity field preserved by approximate projection 50
2.12 Initial vorticity distribution for shear layer problem 51
2.13 Doubly Periodic Shear layer vorticity at $t = 0.5$ on (a) 64×64 grid, and (b) 128×128 grid, and at $t = 1.0$ on (c) 64×64 grid, and (d) 128×128 grid. Note formation of spurious vortices in solution in 64×64 solution. Deformation of the vorticity contours near the edges in the 64×64 solution is an artifact of the contour plotter. 52

3.1 Improper Refinement 61
3.2 Illustration of the proper nesting requirement 62
3.3 Interpolation at a coarse-fine interface 66
3.4 Modified interpolation stencil: Since the left coarse cell is covered by a fine grid, use shifted coarse grid stencil (open circles) to get intermediate values (solid circles), then perform final interpolation as before to get "ghost cell" values (circled X's). Note that to perform interpolation for the vertical coarse/fine interface, we will need to shift the coarse stencil down. 66

3.5 Flux register along $\partial\Omega^1$: the dashed lines represent the edge-centered flux register defined along the coarse-fine interface. Note that the flux register has coarse-grid spacing. 70
3.6 Sample one-dimensional coarse-fine interface 76
3.7 Pseudocode description of AMR Poisson multigrid algorithm 81
3.8 Multigrid with $n_{ref} \neq 2$. Because $n^1_{ref} = 4$, we perform an intermediate coarsening in the multigrid cycle before coarsening from level 2 to level 1. 83
3.9 Pseudocode for the conjugate gradient bottom solver 86
3.10 Best Coarsening: Grid configuration at right is the best possible coarsening of the grids at left. 87
3.11 AMR Poisson test problem (a) Source distribution, and (b) Solution 88
3.12 Multigrid Convergence for (a) $n_{ref} = 2$, (b) $n_{ref} = 4$, and (c) $n_{ref} = 8$ 90
3.13 Normalized timings for the Poisson Solver 92
3.14 Bottom-up iteration algorithm 97
3.15 Convergence history – bottom-up iterations 98
3.16 Residual for bottom up iteration: (a) initial residual (after level solves), (b) after 1 multigrid correction iteration, and (c) after 2 multigrid correction iterations 99
3.17 Top-down iteration algorithm 100
3.18 Convergence history – top-down iterations 101
3.19 Residual for top down iteration: (a) initial residual after level solves, (b) after 1 multigrid correction iteration, (c) after 2 multigrid correction iterations, and (d) after 4 multigrid correction iterations. 102
3.20 Errors vs. grid resolution. Errors in (a) L_∞Norm, and (b) L_1 Norm 106
3.21 Effects of local refinement. Errors in (a) L_∞ Norm, and (b) L_1 Norm 108

4.1 Single-grid update for hyperbolic conservation laws 112
4.2 Pseudocode for composite solution advance for two-level case 114
4.3 Coarse and fine fluxes across the coarse-fine interface. 117
4.4 Schematic of subcycled timestep 119
4.5 Pseudocode for recursive timestep used for hyperbolic conservation laws in Berger and Colella 120
4.6 Naive extension of Berger-Colella algorithm to incompressible Euler 127
4.7 Typical synchronization correction, corr. Fine grid is to the right of the coarse-fine interface. 134
4.8 Computing the composite gradient on the coarse side of a coarse-fine interface for cell (i,j), when coarse-fine interface is located at $(i-\frac{1}{2},j)$ edge. Edge-centered gradient at $(i-\frac{1}{2},j)$ is computed by linear extrapolation of edge-centered gradients at $(i+\frac{1}{2},j)$ and $(i+\frac{3}{2},j)$. Edge-centered gradients at $(i+\frac{1}{2},j)$ and $(i-\frac{1}{2},j)$ are averaged to cell center to get $G^{comp}\phi$ at (i,j) 136
4.9 Max(divergence) vs. number of repeated projection applications 138
4.10 Max of the gradient piece returned by the projection vs. number of repeated projection applications. 139
4.11 Recursive level timestep for the incompressible Euler equations. 143
4.12 Velocity predictor portion of level advance algorithm 147
4.13 Synchronization for incompressible Euler equations. 149

4.14 Initialization algorithm for the incompressible Euler equations 154

5.1 Replacing error in fine-grid boundary cells (shaded) with adjacent values 164
5.2 Replacing error on the coarse side of the coarse-fine interface with averaged fine-grid values (shaded). 166
5.3 Richardson Extrapolation time steps 167
5.4 Basic grid generation algorithm 170

6.1 Initial vorticity distribution for single vortex problem. 173
6.2 Initial vorticity distribution for traveling vortex problem. 174
6.3 Initial vorticity distribution for 3-vortex test case 175
6.4 Vorticity distribution for traveling vortex problem after (a) 50 timesteps, (b) 75 timesteps, (c) 100 timesteps, and (d) 150 timesteps. 177
6.5 32×32 single-grid case after 150 timesteps 178
6.6 Λ after (a) 1 timestep, (b) 10 timesteps, and (c) 20 timesteps. Pictures on left are *with* volume-discrepancy correction, pictures on right are without. 179
6.7 Max(Λ) vs. time for the single-vortex case; (a) without volume-discrepancy correction, and (b) with correction. Note that (a) and (b) have different scales. 180
6.8 Λ (without volume-discrepancy correction) after (a) 2, (b) 24, (c) 60, and (d) 100 timesteps. 182
6.9 Λ (with volume-discrepancy correction) after (a) 2, (b) 24, (c) 60, and (d) 100 timesteps. 183
6.10 Max(Λ) vs. time for the traveling vortex pair case; (a) without volume-discrepancy correction, and (b) with correction. Once again, note that (a) and (b) have different scales. 184
6.11 Vorticity and grid configuration for three-vortex case, 64×64 base grid, one $n_{ref} = 2$ refinement. 186
6.12 Error in x-velocity at t=0.128 for (a) 128×128 single-grid computation, (b) 64×64 base grid with one factor 2 refinement, and (c) 32×32 base grid with one factor 4 refinement. 192
6.13 Error in y-velocity at t=0.128 for (a) 128×128 single-grid computation, (b) 64×64 base grid with one factor 2 refinement, and (c) 32×32 base grid with one factor 4 refinement. 193
6.14 Error in x-velocity at t=0.128 for (a) 128×128 single-grid computation, (b) 64×64 base grid with one factor 2 refinement, and (c) 32×32 base grid with one factor 4 refinement. Note that the scale of the color map has been altered to emphasize small errors 194
6.15 Error in y-velocity at t=0.128 for (a) 128×128 single-grid computation, (b) 64×64 base grid with one factor 2 refinement, and (c) 32×32 base grid with one factor 4 refinement. Note that the scale of the color map has been altered to emphasize small errors 195

7.1 Basic class structure for AMR computations. Solid arrows indicate membership, while dashed lines indicate derivation. In this figure, the AMR class contains a generic AMR Level class (actually, an array of them), and the physics-dependent class is derived from the generic Amr Level class. 199
7.2 Class structure for elliptic solver classes. Solid arrows indicate membership, while dashed arrows indicate derivation. 201
7.3 Class structure for time-dependent incompressible Euler classes. Solid arrows indicate membership, while dashed arrows indicate derivation. 202

List of Tables

2.1 Convergence for velocity, time = 0.5 . . . 54
2.2 Convergence for velocity, time = 1.0 . . . 55

3.1 Number of cells at each grid resolution, tabulated for different base grid sizes when solving sample problem . . . 89
3.2 Convergence rates (average factor by which max(residual) is reduced for each multigrid iteration), tabulated for different refinement ratios and number of levels of refinement . . . 91

6.1 Richardson extrapolation error estimator tolerances used for three-vortex problem . 186
6.2 Errors for x-velocity, time = 0.128 . . . 187
6.3 Errors for y-velocity, time = 0.128 . . . 188

Acknowledgments

I would like to gratefully thank everyone who played a part in getting this work done, either directly or indirectly. The list of people who I've relied on is so huge that I'm sure to miss many, so my deepest apologies (and sincerest gratitude) to everyone.

That said, I would like to especially thank my advisor, Dr. Phil Colella, who really was willing to go the extra mile, whether it was taking the time to patiently explain things yet again, using his immense grasp of numerical mathematics to devise a solution to the latest problem, or being patiently supportive when the entire lab took up sports like mountain biking and proceeded to start breaking bones.

I would also like to thank the readers of the thesis, Professors Carey and Demmel, for their careful reading of the document and their many helpful comments and corrections. I really appreciate the effort, given how busy I know you are.

Sharing code with Rick Propp and Matt Bettencourt immensely speeded up the development time for my code, and resulted in a much better piece of work. Without Vince Beckner's AmrVis package, and Hans Johansen's extensions of Allen Downey's VIGL package, I would have, quite literally, been shooting in the dark on many occasions. Also, I would like to thank Ann Almgren for always being willing to provide insight, whether about the algorithmic issues at hand, or larger "Where do I go now" themes.

My parents, who have been immensely supportive, even when it seemed like I was never going to finish, also deserve credit for raising me so that I could get to this point. They probably deserve the most credit, because without them, this work would not exist, and so this represents their accomplishment as well (and it means that their oldest son is *finally* going to get a "real" job...)

And to everyone outside the lab for getting me out and keeping me sane, especially Mary, who put up with me when I was frustrated, tired, preoccupied, and busily trying to get done. Thanks for being so patient – I will make it up to you somehow. Also, thanks to the I-house gang – all of you (even if you didn't really live in I-house). Y'all made grad school the best years of my life (so far, at least).

I would also like to thank the taxpayers of the United States of America, who generously supported this work through the following grants and programs:

- Computational Science Graduate Fellowship, U.S. Department of Energy.
- "Adaptive Numerical Methods for Partial Differential Equations," US Department of Energy Mathematical, Information, and Computational Sciences Division, Grant DE-FG03-94ER25205.
- "Computational Fluid Dynamics and Combustion Dynamics," US Department of Energy High-Performance Computing and Communications Grand Challenge Program, Grant DE-FG03-92ER25140.
- Research supported at the Lawrence Berkeley National Laboratory by the US Department of Energy Mathematical, Information, and Computing Sciences Division under contract DE-AC03-76SF00098.

Chapter 1

Introduction

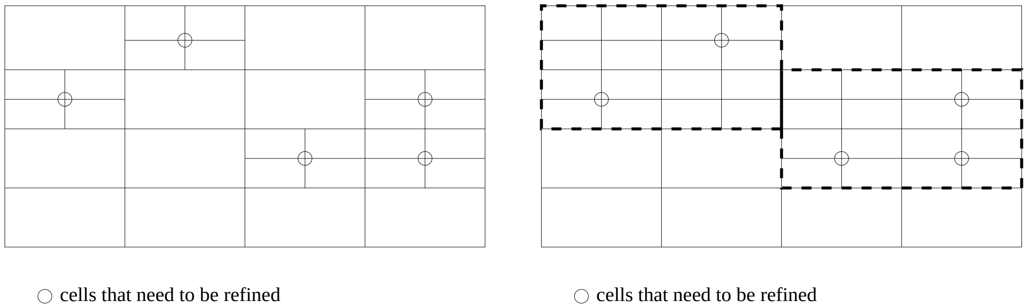
Adaptive numerical methods which focus computational effort where it is needed have been a focus of much research. Solutions to the equations which govern the behavior of many physical phenomena, including those of fluid dynamics, can display behavior over a great number of scales. In many cases, it is necessary to resolve features on very small scales in order to accurately compute larger-scale features of the solution.

The traditional solution to numerical under-resolution of a problem has been to employ a uniformly fine computational mesh over the entire problem. In general, for a finite-difference/finite-volume method, the accuracy of the solution depends on the mesh spacing. The finer the finite-difference mesh (i.e. the more mesh points in a given region), the more accurate the solution. Unfortunately, due to limitations in computational resources, it is often impossible to use a single uniform mesh to solve a given problem to the desired accuracy. On the other hand, it is often the case that the finest resolution is only required in regions which only make up a small fraction of the computational domain. Computing an unnecessarily fine solution outside these regions represents a waste of computational resources. One example of this arises in aerodynamics. When computing the flow around an aircraft, high resolution is often required to adequately resolve rapidly varying features like boundary layers and wakes, while in large regions far from the body, the solution varies little, requiring less resolution to accurately represent the solution. In many cases, use of a uniform fine mesh will result in the bulk of available computational resources being spent computing an unnecessarily accurate solution in regions far from the body, while important flow features near the body are under-resolved due to a lack of resources.

In recognition of this, there has been much effort toward developing methods which adapt the finite-difference mesh to place more grid points in regions where higher resolution is needed, while using fewer grid points in regions where a coarser mesh is sufficient to adequately resolve the solution. Baker [11] provides a good survey of adaptive methods in a finite-element context (although the refinement concepts are generally applicable).

One strategy is known as *clustering*. In this approach, the total number of grid points and grid topology is kept constant, and the grid itself is moved to place higher resolution (in the form of a finer mesh) where necessary, while coarsening the grid in regions where a fine mesh is deemed unnecessary. The mesh itself deforms to follow features in the flow. For this reason, this approach is often known as the *moving-grid* approach. This approach has found the most application in aerodynamics, particularly in steady-state solutions. Advantages of this method include preservation of the basic topology of the mesh, which can be very useful in parallel implementations because partitioning and load-balancing of the solution can be maintained as the solution evolves. Also, it has the advantage of a uniform discretization on a fixed-logic mesh, rather than discrete coarse and fine regions. With the proper grid generation algorithms, transitions between coarse and fine regions can be made smooth, eliminating discontinuities in the computational mesh itself. In principle, the goal of this strategy is equidistribution of error among all the cells in the computational domain for a given number of mesh points. In this sense, solutions are optimal, in that they represent the best possible use of available computation [11]. Like the moving grid approach, unstructured grids also have the advantage of a uniform problem description, in that the discretizations used in refined regions will be the same as those used in coarser regions. This makes it easy to add adaptivity to an existing method, because it does not involve new discretizations. Unfortunately, it is generally more difficult to control the accuracy of the discretization in unstructured-grid methods, particularly for problems without a variational formulation. As in the moving grid approach, care must also be taken to control the quality of the resulting grid, with respect to stability and conditioning of the representation of the problem on the mesh. Moreover, unstructured grid methods generally require more memory to store the various metrics necessary for computation.

Structured meshes, in contrast, are made up of a regular tessellation of cells which all have the same local connectivity. The most common type of structured mesh is a rectangular Cartesian mesh. Design and implementation of finite-difference methods on structured meshes is very well understood, and the regularities of the mesh can be exploited to increase the accuracy of the discretization. Also, it is simple to apply multigrid accelerated iterative methods to construct construct fast elliptic and parabolic solvers for structured meshes. The main disadvantages of structured meshes has been the difficulty of adapting such meshes to complex geometries, although some progress has been made in this area [2, 3, 8, 21, 28, 50]. Also, local refinement on structured meshes will result in a discontinuity in the mesh spacing between coarse and refined cells, which often entails a loss of accuracy. Structured mesh finite-difference approaches have been used extensively in a variety of applications, including aerodynamics, shock dynamics, and atmospheric fluid dynamics.


○ cells that need to be refined
○ cells that need to be refined

(a) (b)

Figure 1.1: Local refinement strategies: (a) cell-by-cell refinement, (b) block-structured refinement

On structured meshes, local refinement can either be cell-by-cell or block-structured (Figure 1.1). In cell-by-cell refinement, cells are individually chosen for refinement and are refined individually. While this is efficient, in that no cells are refined unnecessarily, it leads to fairly complicated tree-like data structures which must maintain nearest-neighbor lists. This can result in a large amount of overhead in managing the AMR computation.

In contrast, we will use block-structured refinement, in which cells tagged for refinement are grouped together into blocks, which are then refined in logically rectangular patches. While this results in some unnecessary refinement, it enables greater efficiency in managing the composite grid structure, since there is a smaller number of irregular nodes (on the order of one per patch, rather than one per node), and irregular indexing is confined to coarse-fine boundaries, rather than potentially every cell. Also, this makes it simpler to separate the implementation of uniform-grid algorithms from the adaptive aspects of the calculation, which in most cases can be represented as boundary conditions on the various refined regions. Because most of the calculation can be done on rectangular arrays of data, it is easier to optimize the bulk of the computation. Advancing blocks of cells also makes it easier to refine in time as well as space, by taking smaller timesteps when advancing refined patches. Unfortunately, this does make the complete adaptive algorithm more complicated to program, because update operations are performed in two steps: regular grid calculations on unions of rectangular grids which make up levels of refinement, and calculations on an irregular set corresponding to the boundary of the union of rectangles at a level of refinement with the grids which make up the coarser levels of refinement. In contrast, when updating the solution on an unstructured grid, only one set of operations must be performed, although the unstructured nature of the mesh requires that they be irregular in nature for all cells in the computational domain.

The refinement strategy we will follow is based on that of Berger and Oliger [19], as extended for hyperbolic conservation laws by Berger and Colella [18]. In [19], regions marked for refinement were covered by rectangular patches of refined cells. These patches could be oriented arbitrarily to better align with solution features like shocks. Along with spatial refinement, their method also included temporal refinement ("subcycling") – refined patches were updated using a smaller timestep than that of the coarse grid. To simplify inter-level communication and boundary conditions, many later implementations based on this strategy (for example that in [18]) did not orient refined patches arbitrarily, instead nesting refined patches completely within coarse grid cells, which aligned the refined grids with the coarse mesh. A variation of this strategy was used by Arney and Flaherty [9] who used tree-structured block refinements of individual patches of cells.

The basic block-structured refinement strategy of [19] has been applied successfully for a number of applications. It was applied to gas dynamics calculations in two dimensions by Berger and Colella [18], and in three dimensions by Bell, et al. [15]. Steinthorsen et al. [58] extended this methodology to the compressible Navier-Stokes equations. Berger and Jameson [21] and Dudek [34] developed methods for computing steady-state compressible flows in complicated geometry using block-structured refinement of mapped grids. Skamarock and Klemp [57] implemented an adaptive method for atmospheric flows based on a compressible flow model which retained the arbitrarily oriented subgrids of [19]. Algorithms to adaptively compute time-dependent solutions to porous media flows were developed by Hornung and Trangenstein [40] and by Propp [51].

To compute steady-state solutions to the incompressible Navier-Stokes equations, Thompson and Ferziger [65] used an adaptive multigrid method based on the adaptive multigrid algorithm originally developed by Brandt [24]. For time-dependent incompressible flows, Howell and Bell [41] and Minion [48] developed adaptive projection methods which did not refine in time but did enforce the divergence constraint on the composite solution across all levels of refinement. In atmospheric modeling, the anelastic equations for atmospheric motion are similar in structure to those for incompressible flow. Clark and Farley [30] and Stevens [59, 60] constructed adaptive projection methods for the anelastic equations which were fully adaptive in time and space, but which did not enforce the incompressibility constraint on the composite grid hierarchy, but instead on a grid-by-grid basis.

Finally, Almgren et al. [5] have developed an adaptive projection method which refines in time as well as space and which enforces the divergence constraint in a composite sense across all refinement levels. That algorithm is the starting point for this work.

1.1 AMR for Incompressible Flows

Our goal will be to extend the block-structured adaptive mesh refinement (AMR) strategies developed for hyperbolic conservation laws by Berger and Colella [18] to the incompressible Euler equations in two dimensions. The addition of local refinement substantially complicates the design

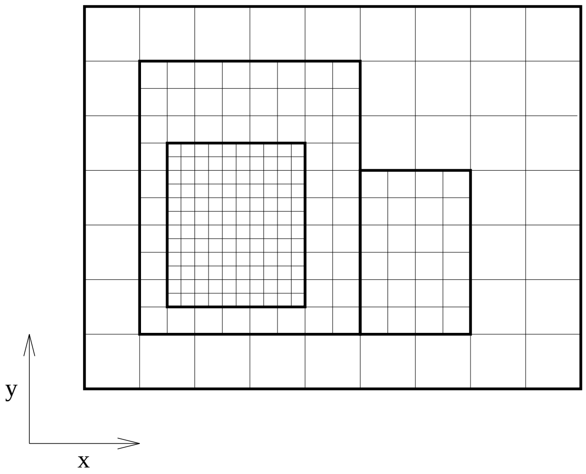


Figure 1.2: Block-structured local refinement. Note that refinement is by a discrete amount and is organized into logically-rectangular patches.

and implementation of projection algorithms, because of the need to sufficiently couple the solutions across interfaces between coarse and fine solutions.

1.1.1 Discretization Issues

Following [18], we will employ nested refinements of block-structured grids along with a corresponding refinement in time as well as space. In this approach, groups of cells are refined in logically rectangular blocks, which simplifies management of refined regions. Additional refinement can easily be nested within existing refined patches, as shown in Figure 1.2.

Using locally refined grids complicates the design of projection methods in many ways. The algorithm of [18] was intended for the solution of hyperbolic conservation laws; conservation was maintained by the use of local corrections where fine and coarse solutions meet. Because solutions to the equations of incompressible flow are also elliptic in nature, additional steps must be taken to ensure that the method presented in this work respects the appropriate smoothness of these solutions in the presence of local refinement.

Special numerical operators must be defined which act on the composite solution across the different levels of refinement. For example, the choice of the discretization of projection operators becomes important. In [41] an idempotent projection discretization was used, in which the finite-difference stencils produced a local decoupling of the computational grid. It was found that this decoupling had to be respected across the interfaces between coarse and fine regions, significantly complicating the algorithm.

We will use a non-idempotent projection algorithm, often referred to as an *approximate projection*, which has simpler stencils. Because repeated application of the approximate projection will not produce the same result as one application, issues like stability of the projection operator and the choice of projecting the velocity field $\mathbf{u}$ or the approximation to $\frac{\partial \mathbf{u}}{\partial t}$ become more important. Almgren et al. [5] use a node-centered projection which was developed using a finite-element formulation; as such, the stability and accuracy of their projection is well understood. We will use the approximate projection of Lai [44], for which there are fewer analytical results, but for which there is also a fairly large body of experience.

Refinement in time as well as space complicates the algorithm as well. Since different regions will be advanced using different timesteps, enforcing the divergence constraint becomes more complicated. For example, if we are projecting the velocity field, $\mathbf{u} \cdot \mathbf{n}$ must be continuous across the interfaces between coarse and fine regions. Since the time-centerings of the velocity on the coarse grid, $\mathbf{u}^c$, and on the fine grid, $\mathbf{u}^f$, can be different due to the different timesteps in coarse and fine regions, enforcing this smoothness becomes more difficult. If the velocity field update $\frac{\partial \mathbf{u}}{\partial t}$ is being projected, enforcing this smoothness becomes more difficult still.

Another issue in the design of locally adaptive methods is that of freestream preservation. In the hyperbolic algorithm of [18], the solution on the coarse grids is updated, and then the solution on the fine grid is updated, using boundary conditions interpolated from surrounding coarse cells. In general, after both coarse and fine advances, the coarse solution in regions covered by refined patches will not be equivalent to the averaged fine solution which covers it. Also, fluxes across the coarse-fine interface computed during the coarse update will not be equal to those computed during the fine update. In order to maintain conservation, the fine solutions are averaged onto the coarse-grid regions, and the flux into the coarse-grid cells adjacent to refined patches is corrected so that the flux into coarse cells across coarse-fine interfaces is the average of the flux computed across the coarse-fine interface from the fine side during the fine-cell updates. These corrections ensure that conserved quantities will be conserved. If the advection scheme is consistent, and there is no explicit space/time dependency of the flux function, then a passively advected scalar field which is spatially constant will remain constant as the flow evolves. This property is known as freestream preservation.

In the case of incompressible flow, the same property of constant scalar fields to remain constant should be observed. For incompressible flow, however, advective fluxes are computed using advection velocities which are themselves computed by solving an elliptic PDE during the local timestep. While averaging fine solutions onto covered regions of coarse grids and correcting advective fluxes into coarse cells adjacent to refined patches (as in [18]) will ensure conservation, there is no guarantee that freestream preservation will be maintained, because there is no guarantee that the coarse- and fine-level advection velocities across the coarse-fine interfaces will be consistent. So, additional steps must be taken to ensure that the property of freestream preservation is maintained in the computation of incompressible flows on locally refined grids.

While the algorithm in Almgren et al. [5] addresses each of these issues, the specifics of the algorithm are complicated. Many specialized algorithmic pieces are required to enforce the appropriate smoothness and conservation of the solution. Our approach has been to attempt to simplify the algorithm to reduce the number of algorithmic components, with the goal of making it easily extensible to more complicated problems.

This work extends [5] in several important ways. First, we employ a cell-centered discrete projection operator, similar to the one developed in [44] and used in [48]. Because our algorithm will require a cell-centered solver for the projection of edge-centered advection velocities and for addition of diffusion (see, for example, [5]), using a cell-centered projection has the advantage of simplicity in that only one set of (cell-centered) solvers need be developed. This will make extension of this work to more complicated problems and geometries much simpler, since the author has found that construction and extension of solvers in a locally adaptive context can be fairly time-consuming. Unlike [48], we refine in time as well as space, which means that, as in [5], a set of synchronization operations must be performed. Also, unlike [48] and [5], we apply the projection to the entire velocity field, instead of to the approximation to $\frac{\partial \mathbf{u}}{\partial t}$, which appears to be necessary for our cell-centered projection when temporal refinement is employed.

Unlike [5], in which levels are synchronized in coarse-fine pairs, we perform multilevel elliptic solves in our synchronization projection, synchronizing all levels which have reached the same point in time. Once a multilevel solver has been developed, this is conceptually simpler, and there is some evidence that synchronization based on multilevel solves are more accurate, at least in the cell-centered case. Finally, we maintain freestream preservation in a different way. In [5], freestream preservation is maintained exactly by a projection of the advection velocity mismatch, followed by a re-advection step on the coarse level. Then, the corrected advective fluxes are interpolated to finer levels. In this work, we instead employ a lagged correction based on the volume-discrepancy formulation used by Acs et al. [1] and by Trangenstein and Bell [66], which is approximate in nature. In practice, we show that the volume-discrepancy method restricts advection errors to those made in the course of a single timestep immediately adjacent to coarse-fine interfaces. In contrast, without this correction, advection errors accumulate and are advected by the flow, corrupting the global solution.

While the algorithm in [5] solves the incompressible Navier-Stokes equations, this work will solve the equations of inviscid incompressible flow, leaving extension to the viscous case for future work. Since the stability issues involved in implementing projection methods for incompressible flow are more difficult in the inviscid case, the inviscid equations are a sufficient test for the new projection discretization and algorithm presented in this work.

1.2 Thesis Overview

In this thesis, a second-order adaptive projection method for the incompressible Euler equations in two dimensions is presented and results are presented to demonstrate its effectiveness.

Chapter 2 describes the single-grid implementation of the projection method we will use. First, the projection method is introduced, and some background is presented, along with the construction of the various discretizations of the projection used in this work. Then, our single-grid strategy of point relaxation coupled with multigrid acceleration for solving Poisson's equation is described and explained. Finally, the single-grid version of the projection algorithm used in this work is outlined and its convergence is demonstrated .

In Chapter 3, our adaptive strategy for numerically solving Poisson's equation on an adaptive hierarchy of refined grids is presented. Special care is given to the issues of coarse-fine matching conditions and the construction of composite operators. This solution strategy is then extended to the special cases of solving on individual levels and groups of levels which do not comprise the entire computational domain. Because the solution of Poisson's equation is central to the the projection method, we also explore other strategies for solving the equations in the context of AMR, with an eye toward the eventual implementation in the time-dependent adaptive projection method. Finally, an error analysis of the adaptive solutions is performed, to determine the sources and size of the errors in this method.

In Chapter 4, the single-grid projection algorithm of Chapter 2 is extended to an adaptive framework. First, the issues raised by the addition of local temporal and spatial refinement are explored, including subcycling (refinement in time), coarse-fine boundary conditions, the construction of composite and level-based operators, and the synchronization operations which must be performed to maintain the proper smoothness of the solution for conservation and accuracy. Then the recursive timestep on a level in the multilevel algorithm is presented. Finally, the method used to initialize the computation at the initial timestep and after a regridding operation is outlined.

Chapter 5 describes the various methods used to estimate the error in a solution in order to decide where to place refined patches. In this work, the grids can be pre-defined by the user, the user can provide a solution-based error criteria, or Richardson extrapolation can be used to estimate the local truncation error of the solution to determine which cells to refine. Also, the method used to group the “tagged” cells into clusters and create a new grid hierarchy is described.

Results of test problems used to validate the method are presented in Chapter 6. Several test problems were selected to demonstrate the convergence and performance of the method. In particular, we demonstrate that flow features are not significantly corrupted by passage through a coarse-fine interface, that the volume-discrepancy correction is an effective tool for controlling advection errors, and that the use of local refinement with this algorithm allows attainment of the accuracy of the equivalent fine-grid computation.

In Chapter 7, issues involved in designing and implementing the adaptive algorithm in software form are discussed. Extensive use was made of the object-oriented functionality of the C++ programming language, along with the numerical optimization of FORTRAN 77. Also, we used BoxLib [52], a C++ class library designed to assist in the implementation of finite-difference on logically rectangular grids.

In the final chapter, we summarize what was learned in the course of this work and present conclusions based on the final results.

Chapter 2

Finite Difference Methods on a Single Grid

This chapter will describe the single-grid projection method used to solve the Euler equations in this work.

2.1 Finite-Difference Notation

In a finite-difference computation, the spatial domain is discretized into a finite number of cells; the goal of the computation is to approximate the exact solution in each cell. In a consistent and stable computation, the more cells in the computation, the better the approximation. A complete description of the theory and practice of finite-difference methods and their place in the greater framework of numerical mathematics is obviously far beyond the scope of this work; the reader is referred to a basic text in numerical PDE's, such as [62].

We will use a regular structured Cartesian mesh in this work. The two-dimensional domain Ω is divided into cells by placing a regular rectangular grid over the domain. For a rectangular domain extending from (x_{lo}, y_{lo}) to (x_{hi}, y_{hi}), this is simple – in two dimensions, there will be n_x

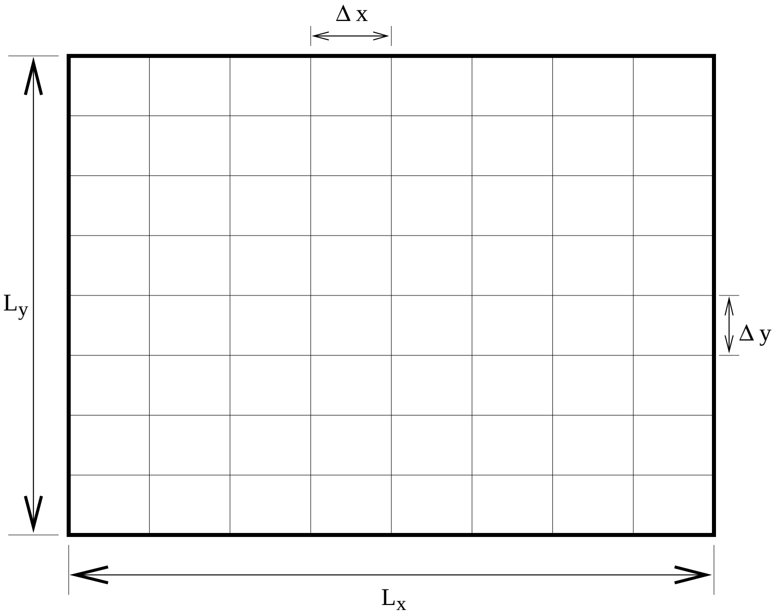


Figure 2.1: Basic Cartesian finite-difference grid. Note that $\Delta x \neq \Delta y$ in this case.

cells in the x-direction, and n_y cells in the y-direction. In general, we will index the cells by $(i, j) = (0..n_x - 1, 0..n_y - 1)$. The cell spacing is denoted as $(\Delta x, \Delta y) = (\frac{L_x}{n_x}, \frac{L_y}{n_y})$, where $L_x = (x_{hi} - x_{lo})$ and $L_y = (y_{hi} - y_{lo})$. See Figure 2.1.

The boundary of Ω will be denoted as $\partial\Omega$; the boundary of the physical domain will also be referred to as a *physical* boundary.

Quantities defined on this Cartesian grid may be cell-centered, edge-centered, or node-centered. A quantity is said to be *cell-centered* if it is defined at the center of the finite-difference cell; $\phi_{ij} = \phi(x_i^{cell}, y_j^{cell})$, where

$$(x_i^{cell}, y_j^{cell}) = (x_{lo} + (i + \frac{1}{2})\Delta x, y_{lo} + (j + \frac{1}{2})\Delta y).$$

A quantity is said to be *node-centered* if it is centered at a node on the computational grid:

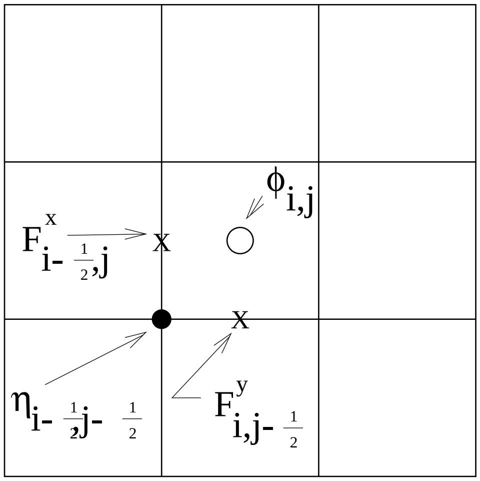


Figure 2.2: Centering of finite-difference quantities. ϕ is cell-centered, η is node-centered, and F^x and F^y are edge-centered

$\eta_{i-\frac{1}{2},j-\frac{1}{2}} = \eta(x^{node}_{i-\frac{1}{2}}, y^{node}_{j-\frac{1}{2}})$, where

$$(x^{node}_{i-\frac{1}{2}}, y^{node}_{j-\frac{1}{2}}) = (x_{lo} + i\Delta x, y_{lo} + j\Delta y).$$

Quantities which are centered along the edges between cells are called *edge-centered.* In two dimensions, an edge-centered quantity is either centered on an x-edge, where it is centered between the (i, j) and $(i-1, j)$ cells on the grid, or it is centered on a y-edge between the (i, j) and $(i, j-1)$ cells. In other words, the x-edge quantity $F^x_{i-\frac{1}{2},j}$ is centered at $(x^{node}_{i-\frac{1}{2}}, y^{cell}_j)$, while the y-edge quantity $F^y_{i,j-\frac{1}{2}}$ is centered at $(x^{cell}_i, y^{node}_{j-\frac{1}{2}})$. See Figure 2.2.

2.1.1 Ghost Cell Implementation of Physical Boundary Conditions

Boundary conditions will be enforced with the use of *ghost cells*, imaginary cells outside the computational domain which will contain appropriate values for ϕ (see Figure 2.3). This has the advantage of computational simplicity as well as ease of programming, since it is often possible

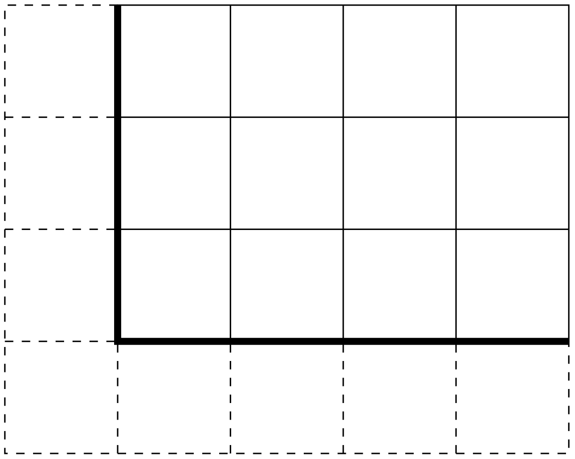


Figure 2.3: Ghost cells outside computational domain. Solid cells are within the computational domain, while dashed cells are ghost cells outside the computational domain used to enforce boundary conditions.

to use the same discretization of the operator both in the interior of the domain and for cells along the boundary. The ghost cell values will be set using appropriate discretizations of the boundary conditions. For Dirichlet boundary conditions, the ghost cell value can be computed as (using the left boundary as an example):

$$\phi_{-1,j} = 2\phi_{BC} - \phi_{0,j}. \tag{2.1}$$

where ϕ_{BC} is the value of $\phi(x_{lo}, j)$ specified for the inhomogeneous boundary condition. This is the result of a linear extrapolation of ϕ through ϕ_{BC} on the physical boundary, and so is $O(h^2)$ for the value of $\phi(-\frac{3}{2}\Delta x, y_j)$. For homogeneous Dirichlet boundary conditions, this reduces to:

$$\phi_{-1,j} = -\phi_{0,j} \tag{2.2}$$

We can also represent an inhomogeneous Dirichlet boundary condition by the third-order extrapolation formula:

$$\phi_{-1,j} = \frac{8}{3}\phi_{BC} - 2\phi_{0,j} + \frac{1}{3}\phi_{1,j}. \tag{2.3}$$

For Neumann boundary conditions, the ghost-cell value is given by:

$$\phi_{-1,j} = \phi_{0,j} - \Delta x \Big(\frac{\partial \phi}{\partial n}\Big)_{BC} \tag{2.4}$$

where $\frac{\partial \phi}{\partial n}_{BC}$ is the normal derivative specified by the Neumann boundary condition. For homogeneous Neumann boundary conditions, $\left(\frac{\partial \phi}{\partial n}\right)_{BC}$ is 0. Finally, at times we will want to extrapolate to compute boundary conditions. We will use either linear extrapolation:

$$\phi_{-1,j} = 2\phi_{0,j} - \phi_{1,j} \tag{2.5}$$

or second-order extrapolation:

$$\phi_{-1,j} = 3\phi_{0,j} - 3\phi_{1,j} + \phi_{2,j}. \tag{2.6}$$

2.2 Poisson's Equation

Poisson's equation appears in the descriptions of many physical problems, such as fluid dynamics, and electrodynamics, and will be necessary for our solution algorithm for the incompressible Euler equations. Because of the simplicity of our problem (for the constant-density Euler equations we will be restricted to the constant-coefficient case), we will be able to employ fairly simple solution techniques.

We wish to solve the constant-coefficient Poisson's equation:

$$\Delta\varphi = \nabla \cdot \nabla\varphi = \rho \quad \text{on } \Omega, \tag{2.7}$$

with boundary conditions:

$$a(x,y)\frac{\partial \varphi}{\partial \mathbf{n}} + b(x,y)\varphi = f(x,y) \quad \text{on} \quad \partial\Omega. \tag{2.8}$$

Note that if a is zero, we are solving a problem with Dirichlet boundary conditions, while if b is zero we are solving a problem subject to Neumann boundary conditions.

For simplicity, we will assume a uniform Cartesian grid in two dimensions, with $\Delta x = \Delta y = h$. The extension to a non-uniform mesh is straightforward. (See, for example, [36].) $\phi_{i,j}$ will be the discrete approximation to $\varphi(x_i, y_j)$. ϕ and ρ will be cell-centered, so that $\phi_{i,j}$ represents the solution at $(x, y) = (h(i + \frac{1}{2}), h(j + \frac{1}{2}))$. We will use the standard 5-point discretization of the Laplacian operator:

$$(L\phi)_{i,j} = \frac{(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j})}{h^2} \tag{2.9}$$

For the time being, we will restrict our discussion to homogeneous Dirichlet boundary conditions, $\varphi = 0$ on $\partial\Omega$; this corresponds to the boundary condition (2.8) with $a = 0$, $b = 1$, and $f = 0$. The physical boundary conditions will be enforced using the ghost-cell formalism described in Section 2.1.1, and we will approximate these boundary conditions using (2.1). We will use multigrid accelerated point relaxation to solve this equation, because the extension of multigrid to the locally-refined case is straightforward and well-understood.

2.2.1 Truncation Error Analysis

We define $\phi^e_{i,j}$ as the exact solution to the continuous problem, evaluated at the cell-centers:

$$\phi^e_{i,j} = \varphi(x_i, y_j) \tag{2.10}$$

Then the truncation error $\tau_{i,j}$ is defined as:

$$\tau_{i,j} = \rho_{i,j} - L(\phi^e)_{i,j} \tag{2.11}$$

where L is the discretization of the Laplacian operator given in (2.9).

It is not difficult to show, using Taylor expansions, that

$$\tau_{i,j} = \begin{cases} O(h^2) & \text{for interior cells} \\ O(1) & \text{for boundary cells: } i = 0,\ i = (n_x - 1),\ j = 0, \text{ or } j = (n_y - 1) \end{cases} \tag{2.12}$$

The truncation on the interiors is $O(h^2)$ due to a cancellation of errors inherent in the centered-difference discretization of L. As we see, there is a loss of accuracy at the boundary, in part because we can no longer take advantage of these cancellations, and in part because of the lower-order nature of the boundary condition discretization in (2.2).

If we define the solution error ξ,

$$\xi_{i,j} = \phi_{i,j} - \phi^e_{i,j} \tag{2.13}$$

then the solution error satisfies the following error equation:

$$L\xi = \tau. \tag{2.14}$$

The solution error ξ is $O(h^2)$ at all points on the grid, despite the lower-order approximation at the boundary. This is because, for smooth solutions, it is possible to maintain global accuracy even when using a less-accurate discretization on a set of cells which has a lower dimension than the problem space. In our case, the problem space is two-dimensional, while the reduced-accuracy discretization on the boundary is on a one-dimensional set of cells. In general, we can lose one order of accuracy on a set of cells one dimension less than the problem space. This would imply that we can have an $O(h)$ boundary condition and still maintain global second-order accuracy in our problem. This can be most easily understood using the modified-equation analysis of Johansen [43].

In that approach, we view the discrete solution as the solution to a continuous problem with a piecewise-constant charge distribution in each cell. Since Poisson's equation is linear, we expect that we can separate the solution error ξ into the sum of contributions to the total error from the source in each cell. We define $\xi^{(kl)}$ as the solution error induced by $\tau^{(kl)}$, which is an approximation to the truncation error $\tau_{k,l}$ integrated over the cell volume:

$$\begin{aligned} \xi^{(kl)} &= (L^h)^{-1}\tau^{(kl)}, \\ \tau^{(kl)}_{i,j} &= \begin{cases} h^2\tau_{i,j} & \text{if } (i,j)=(k,l) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2.15}$$

Given the interpretation of $\tau^{(kl)}$ as a charge of strength $h^2\tau_{k,l}$ located at (x_k, y_l), we expect that

$$\xi^{(kl)} = O(h^2)\tau_{k,l}. \tag{2.16}$$

The total error at point (i,j), $\xi_{i,j}$, would then be the sum of the errors induced by the truncation error in each cell in the domain:

$$\xi_{i,j} = \sum_{k,l\in\Omega} \xi^{(kl)}_{i,j}. \tag{2.17}$$

In interior cells, this would be $\tau_{i,j} \times h^2 = O(h^4)$. If the boundary condition discretization is $O(h)$, then in the boundary cells, this would be $\tau^{bndry}_{i,j} \times h^2 = O(h^3)$. There are $O(\frac{1}{h^2})$ interior cells, for a total contribution of $O(h^2)$ to ξ, while there are only $O(\frac{1}{h})$ boundary cells, so their contribution to ξ is also $O(h^2)$. So, using boundary conditions for which $\tau = O(h)$ on the physical boundary still leads to a second-order accurate method.

For the case of Dirichlet boundary conditions, one obtains a sharper result: if cell (k,l) is adjacent to a physical boundary, then $\xi^{(kl)} = O(h^3)\tau^{k,l}$. To leading order in h, the field induced by $\tau^{(k,l)}$ is that induced by a charge $\tau_{k,l}h^2$, plus that induced by an image charge of strength $\tau_{k,l}h^2$, centered at the ghost-cell $(-1,j)$ (for the left boundary). From potential theory, the effect on the solution of a dipole source can be approximated by:

$$\begin{aligned} \xi(d) &\approx \tau_{i,j} \times h^2 \times [ln(d+\Delta x) - ln(d-\Delta x)] \\ &\approx O(h^3) \times O(h). \end{aligned} \tag{2.18}$$

Once again, there are $O(\frac{1}{h})$ boundary cells, so the effect of the boundary cells on the solution error is actually $O(h^3)$, one order less than that of the interior cells. From this, it follows that one can use a boundary condition for which $\tau = O(1)$ at boundary cells, but still maintain second-order accuracy in the solution. In particular, the boundary condition represented by (2.2) leads to a second-order accurate method.

2.2.2 Point Relaxation

It is well known and documented in the literature that solving the resulting system of equations directly is an $O(N^{\frac{3}{2}})$ operation, where $N = n_x n_y$ is the number of grid points. This is computationally expensive; instead, we will use an iterative scheme. Rather than solve the system of equations exactly, we will compute a series of (hopefully) better approximations to the solution, starting with an initial guess ϕ^0, and continuing with the nth iterate $\{\phi^n\}_{n\geq 1}$. We continue until the error in our solution is less than a given tolerance.

One way to construct an iterative method is as an associated unsteady problem for $\tilde{\phi}$, where the steady state solution ($\frac{\partial \tilde{\phi}}{\partial t} = 0$) is the solution:

$$\frac{\partial \tilde{\phi}}{\partial t} = L\tilde{\phi} - \rho \tag{2.19}$$

Defining a step-size λ and using forward Euler, we can discretize this associated problem as:

$$\tilde{\phi}_{i,j}^{n+1} = \tilde{\phi}_{i,j}^{n} + \lambda(L\tilde{\phi}^n - \rho)_{i,j} \tag{2.20}$$

It is obvious that this method is linear, and will also leave the exact solution unchanged.

This still leaves open the question of whether this will converge to a steady-state solution, and if so, how to select the proper λ. One way to analyze this method is to place the problem in residual-correction form. We first define the residual R:

$$Res^n = \rho - L\tilde{\phi}^n \tag{2.21}$$

This is an indication of how well we are solving the equation – at steady state, when we have reached the solution to Poisson's problem, the residual will go to zero. We can also define δ^n as the error in the nth iterate of ϕ:

$$\delta^n_{i,j} = \tilde{\phi}^n_{i,j} - \phi. \tag{2.22}$$

Using equations (2.21) and (2.22) in (2.19), we see that the error satisfies the heat equation:

$$\frac{\partial \delta}{\partial t} = L\delta. \tag{2.23}$$

We would like to see how δ behaves. For stability and convergence we desire that $\delta \to 0$ as $t \to \infty$. In the case of (2.20), we would like to show that for the operator L:

$$L\delta = (I + \lambda\Delta)\delta, \tag{2.24}$$

(where I is the identity operator) that the norm of δ is reduced:

$$||L\delta|| \leq ||\delta||. \tag{2.25}$$

In the case of doubly periodic boundary conditions, we can use Fourier analysis to examine the behavior of δ. To avoid confusion with $i = \sqrt{-1}$, we will temporarily replace the indices (i, j) with (j_x, j_y). Using the discrete Fourier transform:

$$\delta^n_{j_x,j_y} = \sum_{k_x=-\frac{n_x}{2}+1}^{\frac{n_x}{2}} \sum_{k_y=-\frac{n_y}{2}+1}^{\frac{n_y}{2}} a_{k_x} a_{k_y} e^{2\pi i(k_x j_x + k_y j_y)h} \tag{2.26}$$

If we apply a Fourier transform to the operator L,

$$(L\delta)_{j_x,j_y} = \sum_{k_x=-\frac{n_x}{2}+1}^{\frac{n_x}{2}} \sum_{k_y=-\frac{n_y}{2}+1}^{\frac{n_y}{2}} a_{k_x} a_{k_y} \sigma(k_x, k_y) e^{2\pi i(k_x j_x + k_y j_y)h} \tag{2.27}$$

where the *symbol* $\sigma(k_x, k_y)$ represents the effect of the operator L on the error with a wavenumber (k_x, k_y). For stability, $\sigma(k_x, k_y) \leq 1$ for all wavenumbers k_x and k_y. Note that we can relate the finite-difference notation and Fourier space notation by recognizing that we can define a shift operator S to represent a shift by one on the finite-difference grid. So, if we are considering cell (j_x, j_y), cell $(j_x + 1, j_y)$ will be represented by S_x. Likewise, cell $(j_x, j_y - 1)$ will be represented by $-S_y$. In Fourier space, this becomes a multiplication:

$$\begin{aligned} S_x(\delta^n_{j_x,j_y}) &= \sum_{k_x=-\frac{n_x}{2}+1}^{\frac{n_x}{2}+1} \sum_{k_y=-\frac{n_y}{2}+1}^{\frac{n_y}{2}+1} a_{k_x} a_{k_y} e^{2\pi i(k_x(j_x+1)+k_y j_y)h} \\ &= \sum_{k_x=-\frac{n_x}{2}+1}^{\frac{n_x}{2}+1} \sum_{k_y=-\frac{n_y}{2}+1}^{\frac{n_y}{2}+1} a_{k_x} a_{k_y} e^{2\pi i h k_x} e^{2\pi i(k_x j_x + +k_y j_y)h} \qquad (2.28) \end{aligned}$$

So if we consider a single wavenumber component (k_x, k_y), then the shift operator S_x is just a multiplication by $e^{2\pi i k_x}$. The y-direction shift is similar, as is the negative shift.

To understand the behavior of our operator $L(\delta)$, we will examine its effect on a single wavenumber component of the error. If we use the standard 5-point discretization of the Laplacian, then

$$L(\delta)_{j_x,j_y} = \sum_{k_x=-\frac{n_x}{2}+1}^{\frac{n_x}{2}+1} \sum_{k_y=-\frac{n_y}{2}+1}^{\frac{n_y}{2}+1} a_{k_x} a_{k_y} [1 + \frac{\lambda}{h^2}(e^{2\pi i h k_x} + e^{2\pi i h k_y} + e^{-2\pi i h k_x} + e^{-2\pi h k_y} - 4)] e^{2\pi i(k_x j_x + k_y j_y)h} \tag{2.29}$$

So, the symbol is:

$$\begin{aligned} \sigma(k_x, k_y) &= 1 + \frac{\lambda}{h^2}[e^{2\pi i h k_x} + e^{2\pi i h k_y} + e^{-2\pi i h k_x} + e^{-2\pi h k_y} - 4] \\ &= 1 + \frac{\lambda}{h^2}[2cos(2\pi h k_x) + 2cos(2\pi h k_y) - 4]. \qquad (2.30) \end{aligned}$$

The quantity in brackets will never be positive, so to ensure $|\sigma| \leq 1$ for all (k_x, k_y), we require (looking at the worst case, where $cos(2\pi hk_x) = cos(2\pi hk_y) = -1$):

$$\begin{aligned} 1 + \frac{\lambda}{h^2}(-8) &> -1 \\ \lambda &< \frac{h^2}{4} \end{aligned} \tag{2.31}$$

to ensure that all Fourier modes of the error eventually decay to zero.

Since σ is wavenumber-dependent, different wavenumber components of the error will be damped at different rates. First, look at the slowly varying low-wavenumber components of the error. Performing a Taylor expansion of (2.30) around $(k_x, k_y) = (0, 0)$ yields:

$$\sigma(k_x, k_y)|_{(k_x=0,k_y=0)} \approx 1 - 4\lambda\pi^2(k_x^2 + k_y^2). \tag{2.32}$$

Since $\lambda \leq \frac{h^2}{4} = O(h^2)$, this becomes

$$\sigma(k_x, k_y)|_{(k_x=0,k_y=0)} \approx 1 - Ch^2, \tag{2.33}$$

where C is a constant. So, the low-wavenumber error will decay slowly. On the other hand, if we look at the highest wavenumber present, which is $(k_x, k_y) = (\frac{n_x}{2}, \frac{n_y}{2})$ (remembering that $h = \frac{1}{n_x} = \frac{1}{n_y}$), then $cos(2\pi hk_x) = cos(2\pi hk_y) = -1$. In this case,

$$\sigma(\frac{n_x}{2}, \frac{n_y}{2}) = 1 - \frac{8\lambda}{h^2}. \tag{2.34}$$

If $\lambda = \frac{h^2}{8}$, then the highest wavenumber mode will be damped completely in one iteration. So, this method is extremely efficient at damping high-wavenumber components of the error, while damping lower wavenumber components of the error much more slowly.

Note that in our present scheme, attempting to accelerate convergence to steady state by taking the maximum allowable value for the relaxation parameter, $\lambda = \frac{h^2}{4}$, corresponds to replacing the value of δ_{j_x,j_y} with the average of its four neighbors $\delta_{j_x+1,j_y}, \delta_{j_x-1,j_y}, \delta_{j_x,j_y+1}$, and

+1	-1	+1	-1	+1
-1	+1	-1	+1	-1
+1	-1	+1	-1	+1
-1	+1	-1	+1	-1

Figure 2.4: Residual pattern which causes stalled convergence with point-Jacobi iteration

δ_{j_x,j_y-1}. Unfortunately, straightforward application of this scheme using the 5-point discretization of the Laplacian operator suffers from a local decoupling of solution values, resulting in a lack of convergence for the case of a high-wavenumber error (Figure 2.4) . In this case, the values of +1 and -1 simply flip back and forth for each iteration, and the method does not converge to a steady state solution.

To avoid this problem, we instead use Gauss-Seidel with red-black ordering (GSRB) iteration. Instead of applying point-Jacobi iteration to each cell in turn, we apply two half-step operations. We divide the cells into two groups in a checkerboard manner. First, we relax on the "red" cells (where $i+j$ is even) to get an intermediate value $\delta^{n+\frac{1}{2}}$. Then we relax on the "black" cells (where $i+j$ is odd) using $\delta^{n+\frac{1}{2}}$, (Figure 2.5):

$$\begin{aligned}
\delta^{n+\frac{1}{2}}_{j_x,j_y} &= \begin{cases} \delta^{n}_{j_x,j_y} & \text{if } i+j \text{ odd} \\ \delta^{n}_{j_x,j_y} + \frac{h^2}{4}\left(\frac{\delta^{n}_{j_x+1,j_y}+\delta^{n}_{j_x-1,j_y}+\delta^{n}_{j_x,j_y+1}+\delta^{n}_{j_x,j_y-1}-4\delta^{n}_{j_x,j_y}}{h^2} - Res_{j_x,j_y}\right) & \text{if } i+j \text{ even} \end{cases} \\
\delta^{n+1}_{j_x,j_y} &= \begin{cases} \delta^{n+\frac{1}{2}}_{j_x,j_y} + \frac{h^2}{4}\left(\frac{\delta^{n+\frac{1}{2}}_{j_x+1,j_y}+\delta^{n+\frac{1}{2}}_{j_x-1,j_y}+\delta^{n+\frac{1}{2}}_{j_x,j_y+1}+\delta^{n+\frac{1}{2}}_{j_x,j_y-1}-4\delta^{n+\frac{1}{2}}_{j_x,j_y}}{h^2} - Res_{j_x,j_y}\right) & \text{if } i+j \text{ odd} \\ \delta^{n+\frac{1}{2}}_{j_x,j_y} & \text{if } i+j \text{ even} \end{cases}
\end{aligned} \tag{2.35}$$

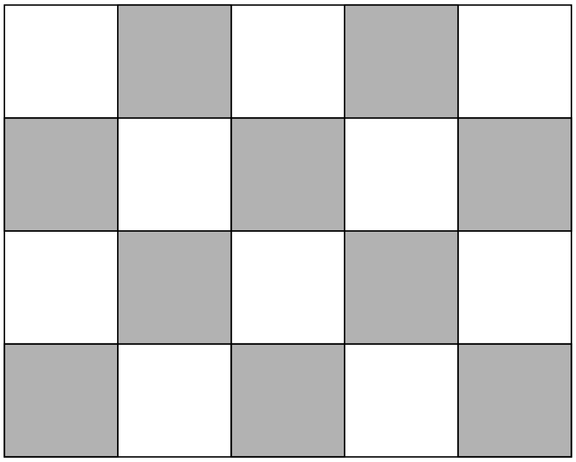


Figure 2.5: "Red-Black" ordering of cells for GSRB iteration

This solves the decoupling problem for the error shown in Figure 2.4, instead replacing it with a uniform (low-wavenumber) error of either 1 or -1.

By itself, use of GSRB is not sufficient to accelerate convergence significantly over point-Jacobi iteration, since for low wavenumber components of the error, $\sigma = 1 - O(h^2)$. However, it is very effective when combined with multigrid acceleration, which is described in the next section.

2.2.3 Multigrid Acceleration

We have seen that GSRB iteration is very effective at damping high-wavenumber components of the error, while it is less effective at reducing lower wavenumber components of the error. In fact, if $\lambda = \frac{h^2}{4}$, high-wavenumber error is replaced by low-wavenumber error by the GSRB iteration. To accelerate convergence, we will employ multigrid acceleration. This technique, originally developed in the 1960s by various researchers [24], has received much attention. A good introductory reference is Briggs [25] or Wesseling [69]. Brandt [24] includes a brief overview of the history of multilevel and multigrid methods. The basic concept is that what constitutes a "high" wavenumber is mesh-dependent. An error mode which is a low-wavenumber error on a fine mesh will be a high-wavenumber error on a coarser mesh. For a given finite-difference grid Ω, we can define a series of coarsened grids Ω^k which cover the domain. In our construction, each successive grid coarsening will be a factor of two coarser than the last, so that $h^{k-1} = 2h^k$. Note that these coarsenings exist independently of any existing AMR grid hierarchy.

The strategy will be to employ a relaxation scheme which effectively damps high-wavenumber components of the error, such as GSRB, on a solution. Then, the solution is restricted to a coarser grid using a restriction operator R_k^{k-1} and relaxed on this grid; applying the relaxation on this mesh will damp a lower wavenumber error, which has become a high wavenumber error on this mesh. This process is continued recursively until a coarsest level is reached, where the problem can be solved inexpensively. Then, the corrections on the coarser levels are interpolated back into the finer level solution with an injection operator I_{k-1}^k, followed by further relaxation on the fine grid to eliminate any high-wavenumber error induced by the interpolation of the coarser corrections. This cycle is then repeated until the residual is sufficiently damped. The multigrid algorithm used in this work is outlined in Figure 2.6. ν_1 is the number of smoothing iterations performed before coarsening, and ν_2 is the number of smoothing operations performed after the interpolation steps.

Because of the simplicity of the problems we are solving, we can use simple averaging for the restriction operator and piecewise constant interpolation as an injection operator. More complicated schemes exist, such as Black Box Multigrid [42], and might be more efficient, but since the simpler methods have worked sufficiently well for our problem, we have not explored these options in this work.

As an example, Figure 2.7 is a plot of residual as a function of multigrid iteration for solving

```
MG()
    $Res^{k^{MAX}} = \rho^{k^{MAX}} - L^{k^{MAX}}(\phi^{k^{MAX}})$
    do while ($iter < maxIter$ and $\|\mathrm{Res}^{\mathrm{k^{MAX}}}\| < \mathrm{tol}$)
        iter++
        MGCycle($\phi^{k^{MAX}}, \rho^{k^{MAX}}$)
        $Res^{k^{MAX}} = \rho^{k^{MAX}} - L^{k^{MAX}}(\phi^{k^{MAX}})$
    end do
end MG

MGCycle ($\phi^k, \rho^k$)
    for $i = 1, \nu_1$
        $\phi^k \leftarrow GSRB(\phi^k, \rho^k)$
    end for
    $Res^k = \rho - L^k(\phi^k)$
    if (k > 0 )
        $\rho^{k-1} = R_k^{k-1}(Res^k)$
        $\delta^{k-1} = 0$
        MGCycle ($\delta^{k-1}, Res^{k-1}$)
        $\phi^k = \phi^k + I_{k-1}^k(\delta^{k-1})$
        for $i = 1, \nu_2$
            $\phi^k \leftarrow GSRB(\phi^k, \rho^k)$
        end for
    else
        BottomSolve ($\phi^0, \rho^0$)
    end if
end MGCycle
```

Figure 2.6: Pseudocode for a multigrid V-cycle

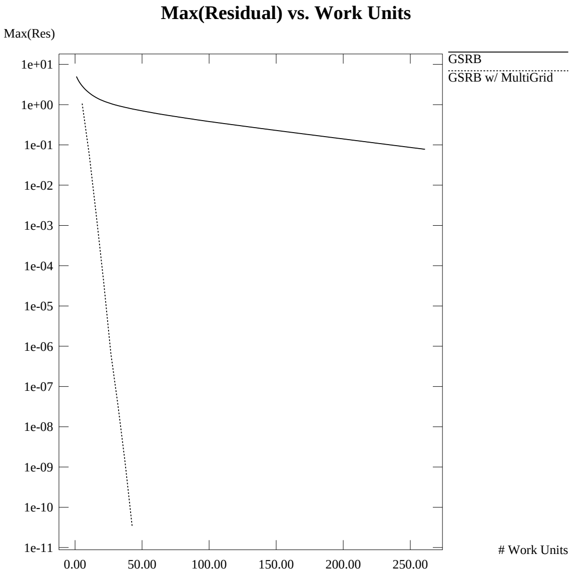


Figure 2.7: Max(Residual) vs. work units for a test problem

a sample problem in which the right hand side is three Gaussian sources (shown in Figure 3.11) with homogeneous Dirichlet boundary conditions, solved on a 32x32 grid. A work unit is the amount of work equivalent to one iteration at the finest level, neglecting interpolation and averaging. The line labeled "GSRB" is a plot of the infinity-norm of the residual as a function of work units when simple GSRB iteration is applied (in the case of GSRB, 1 work unit is 1 GSRB iteration). The line labeled "GSRB w/ Multigrid" shows the maxnorm of residual when GSRB with multigrid acceleration is applied. In this case, 2 GSRB iterations are performed on the way up and 2 more on the way back down on each level. (In the notation of Figure 2.6, this corresponds to $\nu_1 = \nu_2 = 2$.) In this case, each multigrid cycle corresponds to 4 iterations on a 32x32 grid (two on the way down, and two on the way up), 4 iterations on a 16x16 grid, 4 on a 8x8 grid, and so on. So, in this case, each multigrid v-cycle represents 5.33 work units. Even given the added expense of a multigrid cycle over that of simple iteration, it is obvious that multigrid acceleration enables rapid convergence for our test problem.

2.3 The Incompressible Euler Equations

The evolution of inviscid fluid flows can be described by the Euler equations:

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p, && (2.36) \\ \nabla \cdot \mathbf{u} &= -\frac{1}{\rho}\frac{D\rho}{Dt} && (2.37) \end{aligned}$$

where $\mathbf{u}$ is the fluid velocity vector $(u, v)^T$, t is the time, ρ is the fluid density, and p is the pressure. The notation $\frac{D}{Dt}$ represents the material derivative, which in an Eulerian frame of reference is:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + (\mathbf{u} \cdot \nabla). \tag{2.38}$$

The evolution equation for a passively advected scalar in incompressible flow is (we have exploited the fact that $\nabla \cdot \mathbf{u} = 0$):

$$\frac{\partial s}{\partial t} + \nabla \cdot (\mathbf{u}s) = 0 \tag{2.39}$$

For very low Mach number flows, the fluid becomes incompressible, which implies that the material derivative of the density is identically zero. In this case, the conservation of mass equation (2.37) reduces to a constraint on the velocity field,

$$\nabla \cdot \mathbf{u} = 0. \tag{2.40}$$

In our case, we will make the further assumption that the density ρ is constant (extension of this work to the variable density case is straightforward). Since it is a constant, we will without loss of generality assume that $\rho = 1$ identically. The reader is referred to a standard text on fluid mechanics (for example [13]) for the conditions where these approximations are appropriate.

2.4 Projection Methods

While equations (2.36) and (2.40) form a well-posed set of differential equations for the fluid system, it is not clear how to construct a numerical method for their solution. Enforcing the constraint (2.40) will be problematic, and there is no evolution equation for the pressure.

The projection method, originally conceived by Chorin [29] provides a way to evolve a solution to the incompressible Euler equations in time. It is based on the Hodge-Helmholtz decomposition, which uniquely decomposes any vector field $\mathbf{w}$ into a divergence-free part, $\mathbf{u}_d$, and the gradient of a scalar $\nabla\phi$:

$$\begin{aligned} \mathbf{w} &= \mathbf{u}_d + \nabla\phi. \\ \text{where } \mathbf{u}_\mathrm{d}\cdot\mathbf{n} &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{2.41}$$

This is an orthogonal decomposition. If we define the inner product:

$$\langle \mathbf{u}_d, \nabla\phi\rangle = \int_\Omega \mathbf{u}_d \cdot \nabla\phi dV, \tag{2.42}$$

then $\langle \mathbf{u}_d, \nabla\phi\rangle = 0$.

The decomposition can be performed by solving for ϕ in the following partial differential equation:

$$\begin{aligned} \nabla\cdot\nabla\phi &= \nabla\cdot\mathbf{w} \\ \nabla\phi\cdot\mathbf{n} &= \mathbf{w}\cdot\mathbf{n} \quad \text{on } \partial\Omega. \end{aligned} \tag{2.43}$$

Once we have solved for ϕ, then we can extract $\mathbf{u}_d$ by subtracting the gradient of ϕ from the original vector field:

$$\mathbf{u}_d = \mathbf{w} - \nabla\phi. \tag{2.44}$$

We can now define a "projection" operator P, which given a vector field will return the divergence free piece $\mathbf{u}_d$:

$$P(\mathbf{w}) = \mathbf{u}_d. \tag{2.45}$$

To obtain the gradient piece,

$$(I - P)(\mathbf{w}) = \nabla\phi, \tag{2.46}$$

where I is the identity. From equations (2.43) and (2.44),

$$P(\mathbf{w}) = \Big(I - \nabla(\Delta^{-1})\nabla\cdot\Big)\mathbf{w}. \tag{2.47}$$

The projection operator has several important properties. First, it is idempotent, i.e. $P^2(\mathbf{w}) = P(\mathbf{w})$. Second, the projection is symmetric,

$$\langle \mathbf{w}, P(\mathbf{w})\rangle = \langle P(\mathbf{w}), \mathbf{w}\rangle,$$

or $P = P^T$. Also, it is a linear operator, and is norm-reducing, in the sense that $\|P(\mathbf{w})\| \leq \|\mathbf{w}\|$.

If we apply the projection operator to the incompressible Euler equations (2.36) and (2.40), then the divergence constraint (2.40) is no longer a separate equation – the constraint is enforced by the projection. If $\mathbf{u}(t)$ is divergence-free, then $\frac{\partial \mathbf{u}}{\partial t}$ is also divergence-free, in which case, $P(\frac{\partial \mathbf{u}}{\partial t})$ is simply $\frac{\partial \mathbf{u}}{\partial t}$. With this in mind, we can write:

$$\frac{\partial \mathbf{u}}{\partial t} = P\Big(-(\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p\Big). \tag{2.48}$$

We can then discretize this in time as:

$$\begin{aligned} \mathbf{u}^{n+1} &= \mathbf{u}^n - \Delta t P\Big(-(\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p\Big) \\ &= \mathbf{u}^n - \Delta t P\Big(-(\mathbf{u}\cdot\nabla)\mathbf{u}\Big), \end{aligned} \tag{2.49}$$

since $P(\nabla p) = 0$. From this, we can infer that the pressure is not an independent variable, but rather is determined by the requirements of the divergence constraint. In the context of the Hodge-Helmholtz decomposition (2.41), when we project the update to the velocity field in (2.49), the gradient of the pressure and the $\nabla\phi$ term in (2.41) are the same. This is the basis for the projection method.

As formulated by Chorin [29] and extended by Bell, Colella, and Glaz [16], the projection method is a predictor-corrector method which predicts an intermediate velocity field $\mathbf{u}^*$, which is a first approximation of $\mathbf{u}^{n+1}$. The intermediate velocity $\mathbf{u}^*$ is then "projected" onto the space of divergence-free vectors to produce $\mathbf{u}^{n+1}$.

$$\begin{aligned}
\mathbf{u}^* &= \mathbf{u}^n - \Delta t(\mathbf{u}\cdot\nabla)\mathbf{u} \qquad (2.50)\\
\mathbf{u}^{n+1} &= P(\mathbf{u}^*)\\
\nabla p &= (I-P)\mathbf{u}^*
\end{aligned}$$

In this basic example, we have described how a projection method for incompressible flow can be formulated. Note that the details of the spatial and temporal discretizations are left unspecified. There are many different versions of the original projection method in use, with a number of discretization and algorithmic choices. These will be discussed in more depth in subsequent sections.

The basic issues which must be dealt with are the discretization of the projection operator P and the discretization of $(\mathbf{u}\cdot\nabla)\mathbf{u}$ used in the predictor step. We will use the approximate projection of Lai [44], because of its simplified linear algebra. To compute the advective terms, we will use the formulation of Bell, Colella, and Glaz [16], as extended by Bell, Colella, and Howell [17], which uses a lagged pressure gradient, and upwinding in the velocity predictor to achieve second-order accuracy in time.

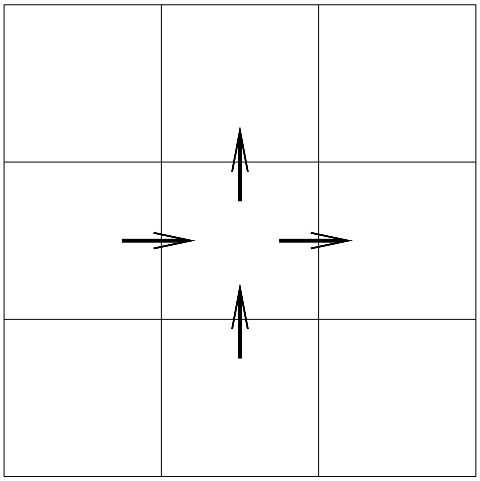


Figure 2.8: Edge-centered velocity field

2.5 Discretizing the Hodge-Helmholtz Projection

In the previous section, the basic projection method proposed by Chorin [29] was outlined, while the details of the spatial discretization was left unspecified. In this section, we will outline the approach we will take in constructing a projection method which we will later extend to an AMR algorithm.

2.5.1 The Discrete Projection

One possible discretization, known as the MAC (for Marker and Cell) [38] projection, is a logical result of a edge-centered velocity field $\mathbf{u}^{edge}$, in which normal velocities are defined at each cell edge (see Figure 2.8). For this reason, we will also refer to the MAC discretization as an edge-centered discretization. Given a velocity field defined at cell edges, we define the cell-centered discrete divergence as:

$$(D\mathbf{u})_{i,j} = \frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{\Delta x} + \frac{v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}}}{\Delta y}. \tag{2.51}$$

This discretization is incomplete without specification of the physical boundary conditions. The boundary condition on the edge-centered velocity field will be the physical boundary condition on the normal velocity at $\partial\Omega$. In the case of solid walls, this will be $\mathbf{u}\cdot\mathbf{n} = 0$ on $\partial\Omega$.

We can also define an edge-centered discrete gradient formula operating on a cell centered variable ϕ as:

$$\begin{aligned}(G\phi)_{i+\frac{1}{2},j} &= \left(\frac{\phi_{i+1,j}-\phi_{i,j}}{\Delta x}, \frac{\phi_{i+1,j+1}+\phi_{i-1,j+1}-\phi_{i+1,j-1}-\phi_{i-1,j-1}}{4\Delta y}\right)^T \\ (G\phi)_{i,j+\frac{1}{2}} &= \left(\frac{\phi_{i+1,j+1}+\phi_{i+1,j-1}-\phi_{i-1,j+1}-\phi_{i-1,j-1}}{4\Delta x}, \frac{\phi_{i,j+1}-\phi_{i,j}}{\Delta y}\right)^T.\end{aligned} \tag{2.52}$$

Following the approach in [44], the boundary condition on the gradient, regardless of the physical boundary condition, is quadratic extrapolation of ϕ, which is equivalent to linear extrapolation of $G\phi$.

In this case, both D and G are second-order centered-difference operators. The operator $L = DG$ is the standard five-point Laplacian operator, and the application of the projection is equally straightforward. First, solve

$$L\phi = D\mathbf{u}^{edge} \tag{2.53}$$

for the cell-centered ϕ. From (2.43), it is apparent that the proper physical boundary condition on ϕ is $\frac{\partial\phi}{\partial n} = \mathbf{w}\cdot\mathbf{n}$. For the case of solid walls, this becomes a homogeneous Neumann boundary condition. So, when solving for ϕ, we will use the ghost-cell implementation of the Neumann boundary condition in (2.4) Further discussion of the boundary conditions for the projection operator is presented in Gresho and Sani [37], and in E and Liu [35]. We then correct the edge-centered velocity field with the gradient of ϕ:

$$\mathbf{u}^{edge} = \mathbf{u}^{edge} - G\phi. \tag{2.54}$$

So, the projection operator looks like:

$$P = I - G(L)^{-1}D. \tag{2.55}$$

While L is not generally invertible, it is invertible on the range of D. This discretization has the advantages of second-order accuracy, and it is simple to implement and generally well-behaved. It is exact in the sense that it maintains in a discrete way the properties of the projection operator outlined in Section 2.4, i.e. $P^2 = P$ and $P = P^T$.

Unfortunately, we would prefer to work with a cell-centered discretization for $\mathbf{u}$ and $(\mathbf{u} \cdot \nabla)\mathbf{u}$ rather than one which is edge-centered. We would like to take advantage of high resolution methods developed for the advection-diffusion equations, which generally require that variables be cell-centered. Also, in the presence of irregular geometries, Cartesian grid methods are much easier to construct with a cell-centered discretization.

2.5.2 Cell-centered Discretization of the Projection

Efforts to formulate an exact discrete projection for cell-centered velocities were largely unsuccessful. We can use the centered-difference operators

$$\begin{aligned}
(D^{CC}\mathbf{u})_{i,j} &= \frac{u_{i+1,j} - u_{i-1,j}}{\Delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{\Delta y} && (2.56)\\
(G^{CC}\phi)_{i,j} &= (\frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x}, \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y})^T. && (2.57)
\end{aligned}$$

Boundary conditions for these cell-centered operators will be similar to those used in the edge-centered case. For the divergence operator, we will use the ghost-cell representation of the physical boundary condition. For solid walls, the homogeneous Dirichlet boundary condition $\mathbf{u} \cdot \mathbf{n}$ is represented using (2.2). Since ϕ is still cell-centered, physical boundary conditions for the gradient will

α	β	α	β	α	β	α	β	α	β
χ	δ	χ	δ	χ	δ	χ	δ	χ	δ
α	β	α	β	α	β	α	β	α	β
χ	δ	χ	δ	χ	δ	χ	δ	χ	δ
α	β	α	β	α	β	α	β	α	β
χ	δ	χ	δ	χ	δ	χ	δ	χ	δ

Figure 2.9: Decoupled grids created by the exact DG operator

be similar to those used in the edge-centered discretization: we will use a quadratic extrapolation of ϕ to fill ghost cells before computing the cell-centered gradient operator.

The corresponding projection, following equations (2.51) through (2.55), is

$$P^{CC} + I - G^{CC}(D^{CC}G^{CC})^{-1}D^{CC}. \tag{2.58}$$

While this discretization produces an idempotent projection, it is not well-behaved. The stencil for $D^{CC}G^{CC}$ produces a decoupling of the grid, in which there are four separate grids which are only coupled together through boundary conditions (Figure 2.9). This decoupling has been shown to cause problems when there are source terms present, and makes implementation of fast linear algebra techniques such as multigrid difficult [44, 55]. Also, Howell and Bell [41] report significant complications when implementing the decoupled stencil across coarse-fine interfaces.

Strikwerda [61] proposed an exact projection which used non-symmetric operators. While this eliminated the decoupling of the grids, it resulted in more complicated linear algebra which proved computationally expensive to implement and suffered from a lack of robustness in the presence of large density gradients [44].

In response to these difficulties, a non-idempotent discretization of the projection was developed by Almgren, Bell, and Szymczak [4]. In this approach, we replace the badly-behaved $D^{CC}G^{CC}$ operator with an approximation to the Laplacian operator L which has better properties. The approximate projection developed in [4] was based on a finite-element formulation, which resulted in a node-centered pressure. The cell-centered discretization we will use was developed by Lai [44]. In this case, we will employ second-order approximations to all the operators involved, using the D^{CC} and G^{CC} from equations (2.56) and (2.57), but replacing the decoupled Laplacian of $D^{CC}G^{CC}$ with a more standard discretization. In our case, we will use the standard five-point Laplacian operator of (2.9), so the projection operator will be:

$$P = I - G^{CC}(L)^{-1}D^{CC} \tag{2.59}$$

Analysis of this projection operator by Lai [44] has shown that it is a second-order accurate operator.

Note that the cell-centered operators can also be constructed with the edge-centered operators using appropriate averaging from edges to cells and from cells to edges. First define the appropriate averaging operators:

$$(Av^{E\rightarrow C}\mathbf{u})_{i,j} = (\frac{u_{i+\frac{1}{2},j} + u_{i-\frac{1}{2},j}}{2}, \frac{v_{i,j+\frac{1}{2}} + v_{i,j-\frac{1}{2}}}{2})^T, \tag{2.60}$$

which averages edge-centered quantities onto cell centers, and

$$\begin{aligned}(Av^{C\rightarrow E}\phi)_{i+\frac{1}{2},j} &= \frac{\phi_{i+1,j} + \phi_{i,j}}{2} \\ (Av^{C\rightarrow E}\phi)_{i,j+\frac{1}{2}} &= \frac{\phi_{i,j+1} + \phi_{i,j}}{2},\end{aligned} \tag{2.61}$$

which averages cell-centered (vector) quantities to cell edges. Then, the cell-centered operators defined in equations (2.56) and (2.57) can be written as:

$$D^{CC}\mathbf{u} = D(Av^{C\rightarrow E}\mathbf{u}) \tag{2.62}$$

and

$$G^{CC}\phi = Av^{E\rightarrow C}G\phi, \tag{2.63}$$

respectively. In this case, we can rewrite the approximate projection (2.59) as:

$$P = I - Av^{E\rightarrow C}G(L)^{-1}DAv^{C\rightarrow E} \tag{2.64}$$

The discretization of (2.64) is not idempotent, $P^2 \neq P$, since $L \neq D^{CC}G^{CC}$. Non-idempotent discretizations of the projection are often referred to as *approximate projections*. Stability and consistency of the approximate projection in (2.64) were shown in [44].

2.5.3 Different Projection Formulations

It was noted in the last section that exact projections for cell-centered variables are not well-behaved, and the approximate projection was introduced as an alternative. In practice, the form of the projection is not the only design choice. We must also determine what is being projected. There are four main variations. For idempotent discretizations of the projection, the different formulations would be equivalent. However, since we are using an approximate projection, choice of the formulation to use has an impact on the performance of the method.

We denote the discrete approximation to the advective term $(\mathbf{u}\cdot\nabla)\mathbf{u}$ as $\mathbf{N^{n+\frac{1}{2}}}$. The first and simplest version of the projection is the *pressure* formulation, in which the projection will return the estimate of the latest pressure, through a projection of the entire velocity field:

1. $\mathbf{u}^{**} = \mathbf{u}^n - \Delta t\mathbf{N^{n+\frac{1}{2}}}$

2. Solve $Lp^{n+\frac{1}{2}} = \frac{1}{\Delta t} D(\mathbf{u}^{**})$
3. $\mathbf{u}^{n+1} = \mathbf{u}^{**} - \Delta t G p^{n+\frac{1}{2}}$.

A variation on the first formulation is known as the *pressure-incremental* formulation, since the projection actually returns an increment to the pressure field, δ^n:

1. $\mathbf{u}^{*} = \mathbf{u}^{n} - \Delta t(\mathbf{N^{n+\frac{1}{2}}} + Gp^{n-\frac{1}{2}})$
2. Solve $L\delta^n = \frac{1}{\Delta t} D(\mathbf{u}^{*})$
3. $\mathbf{u}^{n+1} = \mathbf{u}^{*} - \Delta t G \delta^n$
4. $p^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \delta$.

Note that we discriminate between the use of $\mathbf{u}^*$ and $\mathbf{u}^{**}$ to denote the intermediate velocity field, in an attempt to be consistent with previous definitions of $\mathbf{u}^*$ in the literature, for example, those in [5, 44]. We will use $\mathbf{u}^*$ to refer to the intermediate velocity field which includes the effect of $\nabla p^{n-\frac{1}{2}}$, while we will use $\mathbf{u}^{**}$ to denote the intermediate velocity field computed without the pressure term.

In both of these algorithms, we are projecting a prediction of the new velocity field. If the old velocity $\mathbf{u}^n$ is divergence-free, it is also possible to simply project the update to the velocity field, which is the predicted $\frac{\partial \mathbf{u}}{\partial t}$. As before, this can be done in a pressure formulation:

1. $\mathbf{u}^{**} = \mathbf{u}^{n} - \Delta t \mathbf{N^{n+\frac{1}{2}}}$
2. Solve $Lp^{n+\frac{1}{2}} = D(\frac{\mathbf{u}^{**} - \mathbf{u}^{n}}{\Delta t})$
3. $\mathbf{u}^{n+1} = \mathbf{u}^{**} - \Delta t G p^{n+\frac{1}{2}}$,

or in a pressure-incremental formulation:

1. $\mathbf{u}^* = \mathbf{u}^n - \Delta t(\mathbf{N}^{\mathbf{n}+\frac{1}{2}} + Gp^{n-\frac{1}{2}})$

2. Solve $L\delta^n = D(\frac{\mathbf{u}^*-\mathbf{u}^n}{\Delta t})$

3. $\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t G\delta^n$

4. $p^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \delta^n$.

An analysis of these four formulations is presented by Rider [54], who concluded that for approximate projections, it is better to project the velocity field, rather than $\frac{\partial \mathbf{u}}{\partial t}$, because of the error due to the approximate projection which remains in $\mathbf{u}^n$. This error can accumulate over many timesteps. Also, any errors in the initial state will remain undamped by the schemes which project $\frac{u^*-u^n}{\Delta t}$. However, both the single-grid algorithm of Lai [44], and the AMR algorithm of Almgren, et al. [5] project $\frac{\mathbf{u}^*-\mathbf{u}^n}{\Delta t}$ with success.

Our algorithm will be based on the pressure form of the projection of $\mathbf{u}^{**}$, in large part because of the extra demands of the adaptive algorithm.

2.6 Single-Grid Algorithm

In this section, we will present the single-grid version of the algorithm, which will advance the solution $\mathbf{u}$ and s, where s is a passively advected scalar concentration field, from time t^n to time t^{n+1}. The new pressure $p^{n+\frac{1}{2}}$ will also be computed. It is assumed that at time t^n we have the current solution $\mathbf{u}^n$ and s^n. Our discretization of (2.36) will be:

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \Delta t[(\mathbf{u}\cdot\nabla)\mathbf{u}]^{n+\frac{1}{2}} - \Delta t\nabla\pi^{n+\frac{1}{2}} \tag{2.65}$$

where π will be our approximation of the pressure. We will discretize the scalar update equation as:

$$s^{n+1} = s^n - \Delta t[\nabla \cdot (\mathbf{u}s)]^{n+\frac{1}{2}} \tag{2.66}$$

which will be our evolution equation for s.

The basic structure of the algorithm is a predictor-corrector scheme. We first predict an approximation to the new velocity field, $\mathbf{u}^{**}$. This predicted velocity field will not, in general, satisfy the kinematic constraint on divergence. We then correct the velocity field by projecting $\mathbf{u}^{**}$ onto the space of vector fields which satisfy the divergence constraint.

To compute $\mathbf{u}^{**}$ and s^{n+1}, we will need a set of edge-centered "advection velocities", which are an approximation of $(u^{n+\frac{1}{2}}_{i+\frac{1}{2},j}, v^{n+\frac{1}{2}}_{i,j+\frac{1}{2}})$. To compute these, we predict upwinded approximations for the edge-centered velocities at $t^{n+\frac{1}{2}}$, and then project these velocities using an edge-centered projection, which ensures that our advection velocities are divergence-free. To predict edge values at $t^{n+\frac{1}{2}}$, we use a Taylor series approximation, in which we use (2.36) to replace the $\frac{\partial}{\partial t}$ term. For example:

$$\begin{aligned} u^{L,n+\frac{1}{2}}_{i+\frac{1}{2},j} &= u^n_{i,j} + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\frac{\partial u}{\partial t} \\ &= u^n_{i,j} + \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}[-(\mathbf{u}\cdot\nabla)u - \nabla p]^{n+\frac{1}{2}}_{i,j} \\ &= u^n_{i,j} + \frac{1}{2}(\Delta x - u\Delta t)\frac{\partial u}{\partial x} - \frac{\Delta t}{2}v\frac{\partial u}{\partial y} - \frac{\Delta t}{2}\frac{\partial p}{\partial x} \end{aligned} \tag{2.67}$$

This is an extrapolation from the left side of the edge. In order to compute an upwinded solution at the edge, we will also need an extrapolation from the right, which will look like:

$$\begin{aligned} u^{R,n+\frac{1}{2}}_{i+\frac{1}{2},j} &= u^n_{i+1,j} - \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}\frac{\partial u}{\partial t} \\ &= u^n_{i+1,j} - \frac{\Delta x}{2}\frac{\partial u}{\partial x} + \frac{\Delta t}{2}[(\mathbf{u}\cdot\nabla)u - \nabla p]^{n+\frac{1}{2}}_{i+1,j} \end{aligned} \tag{2.68}$$

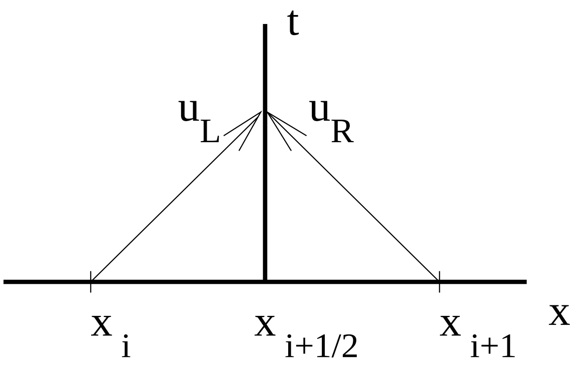


Figure 2.10: Left and right extrapolated states

$$= \quad u^n_{i+1,j} + \frac{1}{2}(-\Delta x - u\Delta t)\frac{\partial u}{\partial x} - \frac{\Delta t}{2}v\frac{\partial u}{\partial y} - \frac{\partial p}{\partial x}$$

See Figure 2.10. In two dimensions, we will also need to compute top and bottom extrapolations for the $(i, j + \frac{1}{2})$ edges. These are computed analogously to the left and right states.

We then use these advection velocities to compute updated values for the scalar field s^{n+1}, and the predicted velocity field $\mathbf{u}^{**}$, which are then projected, completing the update. It is also worth noting that we use convective, rather than conservative, differencing to compute the advection term $(\mathbf{u} \cdot \nabla)\mathbf{u}$ because the advection velocities will not be divergence-free in our multilevel algorithm, although they are solenoidal in this single-grid version.

2.6.1 Computing Advection Velocities

First, we compute approximate edge-centered advection velocities $\mathbf{u}^{edge}$ by averaging the cell-centered $\mathbf{u}^n$ to edges:

$$\mathbf{u}^{edge} = Av^{C\rightarrow E}\mathbf{u}^n. \tag{2.69}$$

Next, we use a Taylor expansion to extrapolate normal velocities to cell edges. For the

$(i+\frac{1}{2},j)$ edges, this will be (using the notation from [5] – the superscript L indicates that the extrapolation to edge $(i+\frac{1}{2},j)$ is from the *left*):

$$u_{i,j}^{norm} = \frac{1}{2}(u_{i+\frac{1}{2},j}^{edge} + u_{i-\frac{1}{2},j}^{edge})$$

$$\tilde{u}_{i+\frac{1}{2},j}^{L,n+\frac{1}{2}} = u_{i,j}^{n} + min[\frac{1}{2}(1 - u_{i,j}^{norm}\frac{\Delta t}{\Delta x}), \frac{1}{2}](u_x)_{i,j} - \frac{\Delta t}{2\Delta y}(\bar{u}_y)_{i,j} \tag{2.70}$$

where u_x is the undivided centered-difference in the normal direction, in this case,

$$(u_x)_{i,j} = \frac{1}{2}(u_{i+1,j}^{n} - u_{i-1,j}^{n}), \tag{2.71}$$

and $\bar{u}_y$ is the undivided upwinded transverse difference:

$$v_{i,j}^{tan} = \frac{1}{2}(v_{i,j+\frac{1}{2}}^{edge} + v_{i,j-\frac{1}{2}}^{edge})$$

$$(\bar{u}_y)_{i,j} = \begin{cases} u_{i,j}^{n} - u_{i,j-1}^{n} & \text{if } v_{i,j}^{tan} > 0 \\ u_{i,j+1}^{n} - u_{i,j}^{n} & \text{if } v_{i,j}^{tan} < 0. \end{cases} \tag{2.72}$$

Computing the right state is similar:

$$\tilde{u}_{i+\frac{1}{2},j}^{R,n+\frac{1}{2}} = u_{i+1,j}^{n} + max[\frac{1}{2}(-1 - u_{i,j}^{norm}\frac{\Delta t}{\Delta x}), -\frac{1}{2}](u_x)_{i+1,j} - \frac{\Delta t}{2\Delta y}(\bar{u}_y)_{i+1,j}. \tag{2.73}$$

Then, we choose the upwind state:

$$u_{i+\frac{1}{2},j}^{n+\frac{1}{2}} = \begin{cases} \tilde{u}_{i+\frac{1}{2},j}^{L,n+\frac{1}{2}} & \text{if } u_{i+\frac{1}{2},j}^{edge} > 0 \\ \tilde{u}_{i+\frac{1}{2},j}^{R,n+\frac{1}{2}} & \text{if } u_{i+\frac{1}{2},j}^{edge} < 0 \\ \frac{1}{2}(\tilde{u}_{i+\frac{1}{2},j}^{L,n+\frac{1}{2}} + \tilde{u}_{i+\frac{1}{2},j}^{R,n+\frac{1}{2}}) & \text{if } u_{i+\frac{1}{2},j}^{edge} = 0 \end{cases} \tag{2.74}$$

Note that we do not include the pressure term in the extrapolation. It is unnecessary because these velocities will be projected. Also, unlike previous implementations of similar algorithms [5, 16, 44] we do not employ slope limiters when computing u_x and u_y. Hilditch [39] found that slope limiters, which were developed to prevent oscillations in compressible flows with sharp discontinuities, are unnecessary for smooth, low Mach number flows.

Extrapolation of normal y-direction velocities is similar. Once we have computed a normal edge-centered velocity field, we apply an edge-centered projection to ensure that our advection velocities will be divergence-free. This is straightforward – we first solve:

$$L\phi = D(\mathbf{u}^{n+\frac{1}{2}}), \tag{2.75}$$

and then correct the velocity field,

$$\mathbf{u}^{AD} = \mathbf{u}^{n+\frac{1}{2}} - G\phi \tag{2.76}$$

We now have a set of edge-centered advection velocities at time $t^{n+\frac{1}{2}}$, which we can use to compute the advective terms in (2.65). Note that we have only computed velocities normal to the faces, which would be $(u^{AD}_{i+\frac{1}{2},j}, v^{AD}_{i,j+\frac{1}{2}})^T$.

2.6.2 Scalar Advection

We also would like to advect a passive scalar concentration field with the flow. Since we have a divergence-free set of edge-centered advection velocities, this is straightforward.

First, we predict edge-centered upwinded values for $s^{n+\frac{1}{2}}$ in the same way as for the velocity predictor. As before, we compute values for $\tilde{s}^{L,n+\frac{1}{2}}$ and $\tilde{s}^{R,n+\frac{1}{2}}$, and then choose the upwind value based on the local sign of $\mathbf{u}^{AD}$.

$$u^{norm}_{i,j} = \frac{1}{2}(u^{edge}_{i+\frac{1}{2},j} + u^{edge}_{i-\frac{1}{2},j})$$

$$\tilde{s}^{L,n+\frac{1}{2}}_{i+\frac{1}{2},j} = s^n_{i,j} + min[\frac{1}{2}(1 - u^{norm}_{i,j}\frac{\Delta t}{\Delta x}), \frac{1}{2}](s_x)_{i,j} - \frac{\Delta t}{2\Delta y}(\bar{s}_y)_{i,j}$$

where, as before,

$$(s_x)_{i,j} = \frac{1}{2}(s^n_{i+1,j} - s^n_{i-1,j})$$

and

$$(\bar{s}_y)_{i,j} = \begin{cases} s^n_{i,j} - s^n_{i,j-1} & \text{if } v^{tan}_{i,j} > 0 \\ s^n_{i,j+1} - s^n_{i,j} & \text{if } v^{tan}_{i,j} < 0. \end{cases} \tag{2.77}$$

For the right state:

$$\tilde{s}^{R,n+\frac{1}{2}}_{i+\frac{1}{2},j} = s^n_{i+1,j} + max[\frac{1}{2}(-1 - u^{norm}_{i,j}\frac{\Delta t}{\Delta x}), -\frac{1}{2}](s_x)_{i+1,j} - \frac{\Delta t}{2\Delta y}(\bar{s}_y)_{i+1,j}.$$

Then, choose the upwind state:

$$s^{n+\frac{1}{2}}_{i+\frac{1}{2},j} = \begin{cases} \tilde{s}^{L,n+\frac{1}{2}}_{i+\frac{1}{2},j} & \text{if } u^{edge}_{i+\frac{1}{2},j} > 0 \\ \tilde{s}^{R,n+\frac{1}{2}}_{i+\frac{1}{2},j} & \text{if } u^{edge}_{i+\frac{1}{2},j} < 0 \\ \frac{1}{2}(\tilde{s}^{L,n+\frac{1}{2}}_{i+\frac{1}{2},j} + \tilde{s}^{R,n+\frac{1}{2}}_{i+\frac{1}{2},j}) & \text{if } u^{edge}_{i+\frac{1}{2},j} = 0 \end{cases} \tag{2.78}$$

Then, we can compute the fluxes:

$$\begin{aligned} F^S_{i+\frac{1}{2},j} &= u^{AD}_{i+\frac{1}{2},j} s^{n+\frac{1}{2}}_{i+\frac{1}{2},j} \\ F^S_{i,j+\frac{1}{2}} &= u^{AD}_{i,j+\frac{1}{2}} s^{n+\frac{1}{2}}_{i,j+\frac{1}{2}}. \end{aligned} \tag{2.79}$$

Finally, the updated state s^{n+1} can be computed using the discrete analog to (2.66):

$$s^{n+1}_{i,j} = s^n_{i,j} - \Delta t(\frac{F^S_{i+\frac{1}{2},j} - F^S_{i-\frac{1}{2},j}}{\Delta x} + \frac{F^S_{i,j+\frac{1}{2}} - F^S_{i,j-\frac{1}{2}}}{\Delta y}). \tag{2.80}$$

2.6.3 Velocity Predictor

Once we have the divergence-free advection velocities, we can compute an approximation of $\mathbf{N}^{n+\frac{1}{2}} = [(\mathbf{u} \cdot \nabla)\mathbf{u}]^{n+\frac{1}{2}}$. Although the advection velocities are discretely divergence-free, which means that conservative differencing could be used to compute $\mathbf{N}^{n+\frac{1}{2}}$, we will instead use convective differencing because in the multilevel case, the advection velocities will not, in general, be discretely divergence-free, for reasons which will be explained in Section 4.4.2.

First, we must re-predict edge-centered velocities as in Section 2.6.1, this time using the projected $\mathbf{u}^{AD}$ rather than $Av^{C\rightarrow E}(\mathbf{u}^n)$, which was used in Section 2.6.1. To save some work, we can re-use the normal velocities $\mathbf{u}^{AD}$ as predicted velocities $\mathbf{u}^{half}$. This means that we only must compute the tangential edge-centered predicted velocities. To compute $v^{half}_{i+\frac{1}{2},j}$, for example, we extrapolate in the same way as we did for $u^{half}_{i+\frac{1}{2},j}$, in this case including $G\phi$ from (2.52) to represent the effects of the edge-centered projection:

$$u^{norm}_{i,j} = \frac{1}{2}(u^{AD}_{i+\frac{1}{2},j} + u^{AD}_{i-\frac{1}{2},j})$$

$$\tilde{v}^{L,n+\frac{1}{2}}_{i+\frac{1}{2},j} = v^n_{i,j} + min[\frac{1}{2}(1 - u^{norm}_{i,j}\frac{\Delta t}{\Delta x}), \frac{1}{2}](v_x)_{i,j} - \frac{\Delta t}{2\Delta y}(\bar{v}_y)_{i,j} \tag{2.81}$$

where

$$(v_x)_{i,j} = \frac{1}{2}(v^n_{i+1,j} - v^n_{i-1,j}), \tag{2.82}$$

and

$$v^{tan}_{i,j} = \frac{1}{2}(v^{AD}_{i,j+\frac{1}{2}} + v^{AD}_{i,j-\frac{1}{2}})$$

$$(\bar{v}_y)_{i,j} = \begin{cases} v^n_{i,j} - v^n_{i,j-1} & \text{if } v^{tan}_{i,j} > 0 \\ v^n_{i,j+1} - v^n_{i,j} & \text{if } v^{tan}_{i,j} < 0. \end{cases} \tag{2.83}$$

For the "right" state,

$$\tilde{v}^{R,n+\frac{1}{2}}_{i+\frac{1}{2},j} = v^n_{i+1,j} + max[\frac{1}{2}(-1 - u^{norm}_{i,j}\frac{\Delta t}{\Delta x}), -\frac{1}{2}](v_x)_{i+1,j} - \frac{\Delta t}{2\Delta y}(\bar{v}_y)_{i+1,j}. \tag{2.84}$$

Then we can choose the upwind state:

$$v^{n+\frac{1}{2}}_{i+\frac{1}{2},j} = \begin{cases} \tilde{v}^{L,n+\frac{1}{2}}_{i+\frac{1}{2},j} & \text{if } u^{AD}_{i+\frac{1}{2},j} > 0 \\ \tilde{v}^{R,n+\frac{1}{2}}_{i+\frac{1}{2},j} & \text{if } u^{AD}_{i+\frac{1}{2},j} < 0 \\ \frac{1}{2}(\tilde{v}^{L,n+\frac{1}{2}}_{i+\frac{1}{2},j} + \tilde{v}^{R,n+\frac{1}{2}}_{i+\frac{1}{2},j}) & \text{if } u^{AD}_{i+\frac{1}{2},j} = 0 \end{cases}. \tag{2.85}$$

Finally, we add the pressure gradient term:

$$\begin{aligned} v^{half}_{i+\frac{1}{2},j} &= v^{half} - G\phi. \qquad (2.86) \\ &= v^{half} - \frac{\phi_{i+1,j+1} + \phi_{i-1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j-1}}{4\Delta y}. \end{aligned}$$

Once we have the edge-centered predicted velocities $\mathbf{u}^{half}$, we can compute the advective terms. First, we compute a cell-centered "advection" velocity $\mathbf{u}^{AD-CC}$:

$$\mathbf{u}^{AD-CC} = Av^{E \to C} \mathbf{u}^{AD}.$$

Then,

$$\begin{aligned} [(\mathbf{u} \cdot \nabla) u]_{i,j}^{n+\frac{1}{2}} &= u_{i,j}^{AD-CC} \frac{(u_{i+\frac{1}{2},j}^{half} - u_{i-\frac{1}{2},j}^{half})}{\Delta x} + v_{i,j}^{AD-CC} \frac{(u_{i,j+\frac{1}{2}}^{half} - u_{i,j-\frac{1}{2}}^{half})}{\Delta y} \\ [(\mathbf{u} \cdot \nabla) v]_{i,j}^{n+\frac{1}{2}} &= u_{i,j}^{AD-CC} \frac{(v_{i+\frac{1}{2},j}^{half} - v_{i-\frac{1}{2},j}^{half})}{\Delta x} + v_{i,j}^{AD-CC} \frac{(v_{i,j+\frac{1}{2}}^{half} - v_{i,j-\frac{1}{2}}^{half})}{\Delta y} \end{aligned} \tag{2.87}$$

Finally, we can now compute $\mathbf{u}^{**}$:

$$\begin{aligned} u_{i,j}^{**} &= u_{i,j}^{n} - \Delta t [(\mathbf{u} \cdot \nabla) u]_{i,j}^{n+\frac{1}{2}} \\ v_{i,j}^{**} &= v_{i,j}^{n} - \Delta t [(\mathbf{u} \cdot \nabla) v]_{i,j}^{n+\frac{1}{2}} \end{aligned} \tag{2.88}$$

2.6.4 Projection

Once we have computed the intermediate velocity $\mathbf{u}^{**}$, all that remains is to project it. Using the approximate projection of Section 2.5.2, this is straightforward. First, we solve

$$L\pi^{n+\frac{1}{2}} = \frac{1}{\Delta t} D^{CC}(\mathbf{u}^{**}) \tag{2.89}$$

using the multigrid accelerated algorithm described in Section 2.2.3. Then, we use this pressure to correct the velocity field onto the space of vector fields which satisfy the divergence constraint:

$$\mathbf{u}^{n+1} = \mathbf{u}^{**} - \Delta t G^{CC} \pi^{n+\frac{1}{2}}. \tag{2.90}$$

2.7 Filters

A discussion of projection methods would be incomplete without mentioning filters. In much of the literature on projection methods, filtering is used to remove non-physical velocity

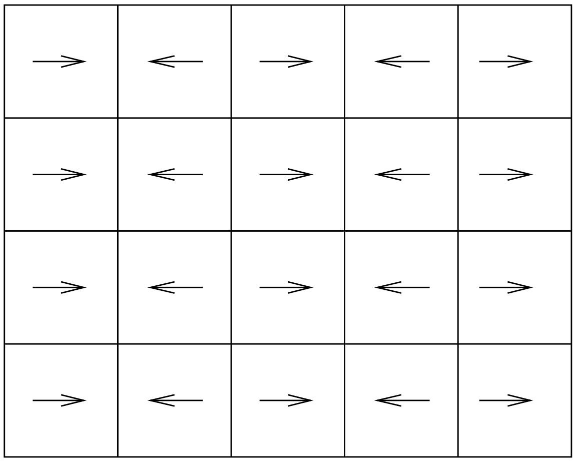


Figure 2.11: Non-physical velocity field preserved by approximate projection

modes that the approximate projection will not remove. As an example, consider the velocity field in Figure 2.11. While it is obvious upon inspection that this velocity field is not solenoidal, it is in the null-space of the D^{CC} operator, so it is not removed by our projection. Lai [44], found that these non-physical modes caused problems with reacting flow. Rider [53] presents a survey of different filter formulations, as well as numerical experiments which point to the necessity of filters for some applications, to remove errors which accumulate and degrade the solution.

While we have implemented the filters mentioned in [53] in our single-grid versions of the code, developing a multilevel filter proved difficult, as will be described in Section 4.7. Also, we saw no apparent degradation in our solution without filters. So, the decision was made to defer filtering until it proved necessary. Since we do not use filtering in our adaptive algorithm, we will not include filtering in our single-grid algorithm either. The reader is referred to [39, 44, 53] for more involved discussions on the role of filtering in projection methods.

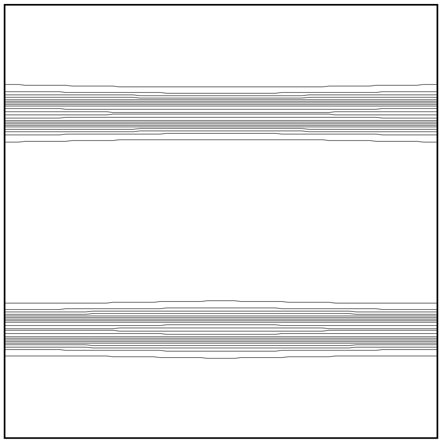


Figure 2.12: Initial vorticity distribution for shear layer problem

2.8 Convergence of the Algorithm

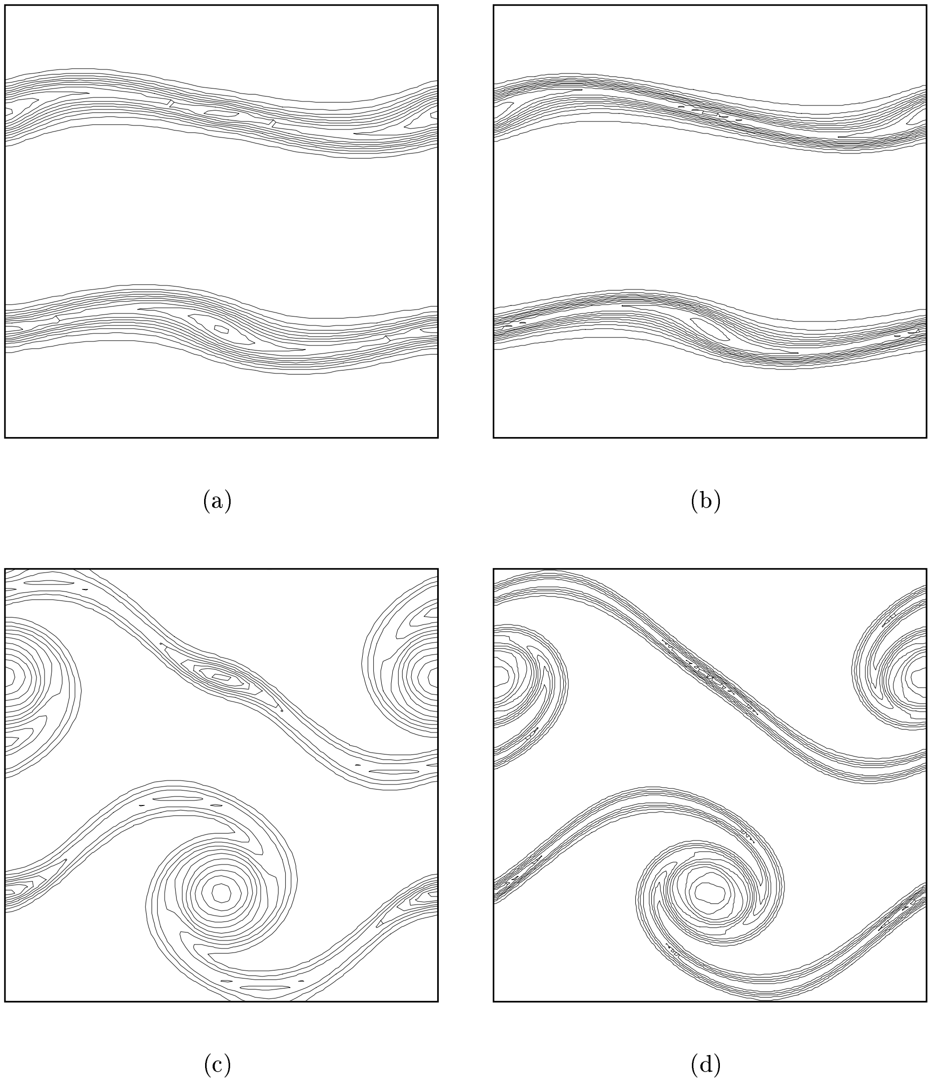
To demonstrate the convergence and accuracy of the single-grid algorithm, solutions to a doubly periodic shear layer were computed on a succession of finer and finer grids. The initial conditions were:

$$\begin{aligned} u(x,y) &= \begin{cases} tanh(\rho_s(y-\frac{1}{4})) & \text{if } y \leq \frac{1}{2} \\ tanh(\rho_s(\frac{3}{4}-y)) & \text{if } y > \frac{1}{2} \end{cases} \\ v(x,y) &= \delta_s sin(2\pi x) \end{aligned} \tag{2.91}$$

with $\rho_s = 42.0$ and $\delta_s = 0.05$. This is the same problem studied by Brown and Minion [26], and represents a shear layer which is between their "thick" and "thin" cases (Figure 2.12).

To test the convergence of the single-grid algorithm, the doubly-periodic vortex test case was run on $32 \times 32, 64 \times 64, 128 \times 128$, and 512×512 grids. Then, the error in each computation was estimated by averaging a finer solution onto the next coarser solution:

$$E^{2h} = Av(\mathbf{u}^h) - \mathbf{u}^{2h} \tag{2.92}$$



(a) (b) (c) (d)

Figure 2.13: Doubly Periodic Shear layer vorticity at $t = 0.5$ on (a) 64×64 grid, and (b) 128×128 grid, and at $t = 1.0$ on (c) 64×64 grid, and (d) 128×128 grid. Note formation of spurious vortices in solution in 64×64 solution. Deformation of the vorticity contours near the edges in the 64×64 solution is an artifact of the contour plotter.

The convergence rate p can then be estimated as:

$$p \approx \frac{1}{ln(2)} ln(\frac{E^{2h}}{E^{h}}) \tag{2.93}$$

The rate of convergence was estimated for both components of velocity. The results are tabulated in Table 2.1, for L_1, L_2, and infinity-norms for time $t = 0.5$ and in Table 2.2 for L_1, L_2, and infinity-norms for time $t = 1.0$. As can be seen, both components of velocity appear to converge at second-order rates. The only exception appears to be the infinity-norm of the $y-$velocity (Table 2.2(c)), which shows a marked degradation in convergence. It is believed that in this case, the appearance of the spurious vortex in the under-resolved cases is affecting these results, since the spurious vortex is present in the coarser cases, but not in the finer cases. While this is not a strong enough effect to be seen in more global L_1 and L_2 norms, it is seen in the local L_∞ norm. For the solution at $t = 0.5$, the spurious flow feature has not yet appeared in a strong enough fashion to affect the convergence results.

	h = $\frac{1}{64} \rightarrow \frac{1}{128}$	$\frac{1}{128} \rightarrow \frac{1}{256}$	$\frac{1}{256} \rightarrow \frac{1}{512}$
Error(u)	0.0107698	0.0026623	0.000559159
Rate	——	2.02	2.25
Error(v)	0.00703588	0.0017225	0.000369297
Rate	——	2.03	2.22

(a) L_1 norm

	h = $\frac{1}{64} \rightarrow \frac{1}{128}$	$\frac{1}{128} \rightarrow \frac{1}{256}$	$\frac{1}{256} \rightarrow \frac{1}{512}$
Error(u)	0.0214418	0.00518821	0.0011087
Rate	——	2.05	2.23
Error(v)	0.00948762	0.00250181	0.000588511
Rate	——	1.92	2.09

(b) L_2 norm

	h = $\frac{1}{64} \rightarrow \frac{1}{128}$	$\frac{1}{128} \rightarrow \frac{1}{256}$	$\frac{1}{256} \rightarrow \frac{1}{512}$
Error(u)	0.105859	0.0282326	0.00760067
Rate	——	1.91	1.89
Error(v)	0.0405071	0.0160055	0.004045
Rate	——	1.34	1.98

(c) L_∞ norm

Table 2.1: Convergence for velocity, time = 0.5

	h = $\frac{1}{64} \to \frac{1}{128}$	$\frac{1}{128} \to \frac{1}{256}$	$\frac{1}{256} \to \frac{1}{512}$
Error(u)	0.0594146	0.0129793	0.00244816
Rate	——	2.19	2.41
Error(v)	0.0549152	0.0136606	0.00250969
Rate	——	2.01	2.44

(a) L_1 norm

	h = $\frac{1}{64} \to \frac{1}{128}$	$\frac{1}{128} \to \frac{1}{256}$	$\frac{1}{256} \to \frac{1}{512}$
Error(u)	0.0842932	0.0211447	0.00473912
Rate	——	2.00	2.16
Error(v)	0.0661976	0.0195002	0.00454566
Rate	——	1.76	2.10

(b) L_2 norm

	h = $\frac{1}{64} \to \frac{1}{128}$	$\frac{1}{128} \to \frac{1}{256}$	$\frac{1}{256} \to \frac{1}{512}$
Error(u)	0.40088	0.133916	0.0273512
Rate	——	1.58	2.29
Error(v)	0.200489	0.0975586	0.0329477
Rate	——	1.04	0.69

(c) L_∞ norm

Table 2.2: Convergence for velocity, time = 1.0

Chapter 3

Adaptive Solutions to Poisson's Equation

This chapter will describe the formulation of a Poisson solver using the cell-centered AMR methodology. Some experimentation was performed to fine-tune the algorithm with an eye toward constructing the projection method we will eventually use to solve the Euler equations. Since projection methods generally involve solving an elliptic equation for the pressure to enforce the divergence constraint, this will be an integral part of the complete adaptive algorithm for the incompressible Euler equations.

3.1 AMR Notation

In this work, we will employ the block-structured local refinement strategy of Berger and Colella [18], in which a hierarchy of nested refinements is employed.

All computations will start with a single base grid, which will be as coarse as possible, in order to best exploit the advantages of adaptivity. This grid will span the entire computational domain, which we will denote as Ω^0. The base grid will have n_x cells in the x-direction and n_y

cells in the y-direction; these cells will be indexed $(i^0, j^0) = (0..n_x^0 - 1, 0..n_y^0 - 1)$, and will have cell spacing $(h_{0,x}, h_{0,y}) = (\frac{L_x}{n_x^0}, \frac{L_y}{n_y^0})$.

Coarse cells will then be tagged for refinement based on some estimate of the error in the solution. (Error estimation will be discussed in more depth in Chapter 5.) Logically rectangular grids of refined cells will then be used to cover these tagged cells. Cells in these refined patches will uniformly be a factor of n_{ref} finer than the cells in the coarse grid, with grid spacing $(h_x^{fine}, h_y^{fine}) = (\frac{1}{n_{ref}} h_{0,x}, \frac{1}{n_{ref}} h_{0,y})$. We will denote the union of these refined grids as a *level*, which is indicative of the fact that all the grids on this level are at a given *level of refinement*.

If additional refinement is necessary, additional refinement can be added by refining patches of the already-refined grids, resulting in a set of still-finer grids nested within the initial refined grids which then make up a new, finer level. This process of nested refinement can be continued until the solution is well resolved in all regions of the domain or a maximum level of refinement is reached. For example, see Figure 1.2, which depicts a sample configuration of nested refinements.

The collection of different levels of refinement makes up a hierarchy of levels. We will index these levels as $\ell = 0..\ell_{max}$, where 0 is the coarsest level (the base grid), and ℓ_{max} is the finest level. Each level $\ell + 1$ will be a factor of n_{ref}^{ℓ} finer in spatial resolution than level ℓ. While in general n_{ref}^{ℓ} could be any integer, we will restrict the refinement ratio to be a power of two to facilitate the use of multigrid acceleration, which was presented in Section 2.2.3. The refinement ratio may vary between different levels; for example, n_{ref}^1 could be 2 while n_{ref}^2 could equal 4.

We will denote by Ω^ℓ the union of grids making up the ℓth level of refinement. In general, Ω^ℓ will be made up of more than one rectangular patch, or grid; these grids will be subregions of Ω^ℓ and will be indexed as $\Omega^{\ell,k}$ where $k = 0...n_{grids}^{\ell} - 1$. So, for example, the level 1 domain Ω^1 will then consist of the union of the level 1 grids, which will contain the cells which have been refined by a factor of n_{ref}^0 from the base level. Note that the grids which make up a level need not be contiguous. In this implementation, grids at the same level of refinement will not overlap, although there is no reason why they could not do so.

We expect that the solution on refined grids is more accurate than the solutions on coarser levels. This leads to the concepts of *valid regions* and the *composite solution.* The refined patches will overlay part of the original coarse Ω^0 grid. We will define the *valid region* of the level 0 grids to be those areas which are not covered by finer grids:

$$\Omega^0_{valid} = \Omega^0 - P(\Omega^1) \tag{3.1}$$

where $P(\Omega^1)$ is the projection of the level 1 grids onto the level 0 grid – the level 0 cells which are covered by level 1. Extending this to the entire multilevel hierarchy of grids, we can say that the valid region will be the union of all cells not covered by refinement:

$$\begin{aligned} \Omega_{valid} &= \bigcup_{\ell=0}^{\ell_{max}} \Omega^\ell - P(\Omega^{\ell+1}) \\ &= \bigcup_{\ell=0}^{\ell_{max}} \Omega^\ell_{valid} \end{aligned} \tag{3.2}$$

In contrast, we will defined the *covered region* as the part of a given level which is covered by a refined grid:

$$\begin{aligned} \Omega^\ell_{covered} &= \Omega^\ell - \Omega^\ell_{valid} \\ &= P(\Omega^{\ell+1}) \end{aligned} \tag{3.3}$$

On each level, the valid region will also contain edges. Because many quantities are edge-centered, we will also need to differentiate between valid and covered edges. On a level ℓ, all edges in Ω^ℓ, whether covered by refinement or not, will be denoted by $\Omega^{\ell,*}$. The *valid edges*, which we will denote by $\Omega^{\ell,*}_{valid}$, will be the cell edges of valid level ℓ cells which are not covered by finer edges. This will consist of all edges of level ℓ cells in the valid region Ω^ℓ_{valid} (including the outer edge $\partial\Omega^\ell$) with the exception of the outer edges of the projection of the next finer level $\ell+1$, which will be considered to be covered by $\partial\Omega^{\ell+1}$, the boundary of $\Omega^{\ell+1}$.

We will define the *composite* solution as the union of solutions on each level's valid region. In other words, the solution will only carry any meaning in the finest cells in a particular location; cells which have been covered by finer grids will not be considered to contain valid information in the composite solution.

However, in many cases, we will want to organize our computations on a level-by-level basis, computing on each grid as if it were a single complete rectangle (to take advantage of vectorization, for example). For this reason, there will often be solution variables which exist for all cells in a given level, regardless of whether they are covered by finer cells or not. The solution defined simply on a level, regardless of whether or not it is covered by a refined patch, will be known as the *level solution*. In many cases, the level solution in regions which are covered by refinement will just be the spatially averaged finer-level solution.

As in Section 2.1, variables can have different centerings. Once again, we can have cell-centered or edge-centered variables, which can then be either composite variables, or level variables. Composite variables are defined over the entire hierarchy of grids, in the valid regions of each grid. On the other hand, level variables exist on each level, in both the valid and covered regions on each grid.

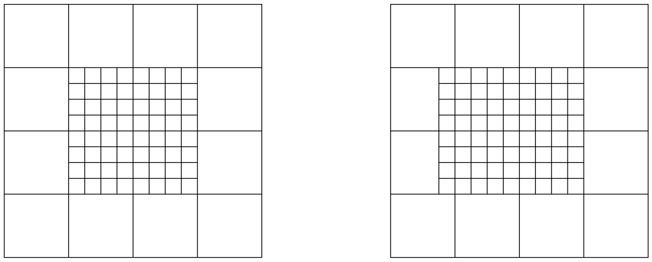
For a cell-centered variable ϕ, the level variable is defined on all of Ω^ℓ, and will be denoted by ϕ^ℓ. Variables which are composite variables, which only have meaning on the union of valid regions on all levels, will be denoted by the superscript *comp*, so ϕ^{comp} is defined on the union of valid regions over all the levels which make up the hierarchy of grids. It will also be useful to define $\phi^{comp,\ell}$, which is the composite variable ϕ^{comp} on the valid region of level ℓ, Ω^ℓ_{valid}. For edge-centered variables, the notation will be similar. In particular, for an edge-centered vector field $\mathbf{F}$, which is defined at normal edges (see Figure 2.8), $\mathbf{F}^\ell$ will be a level variable, defined at all cell edges on level ℓ, $\Omega^{\ell,*}$, while the composite edge-centered field $\mathbf{F}^{comp}$ will be defined on the set of valid edges Ω^*_{valid}:

$$\mathbf{F}^{comp} = \bigcup_{\ell=0}^{\ell_{max}} \mathbf{F}^{comp,\ell}_{valid}.$$

In our refinement scheme, notice that cell edges in covered regions are always overlain by fine-cell edges, in contrast to cell centers. In particular, the edges making up the outer edges of refined grids will overlay the coarse-cell edges which make up the outer edges of the projection of the refined patch. This edge will take on particular importance, because it is the location of the discontinuity in grid spacing. On one side of this edge, the valid solution is on a refined grid; on the other side, the valid solution has coarse-level resolution. For this reason, we will call this the *coarse-fine interface.* We expect that the discontinuity in grid spacing will cause complications in our discretization, and so we expect that in regions neighboring the coarse-fine interfaces special care will be required.

3.1.1 Proper Grid Generation

To simplify boundary conditions and other communication between the solution at different levels of refinement, we will impose two requirements on the multilevel hierarchy of grids. First, we will require that any coarse cell undergoing refinement be refined completely; partial refinement as



Proper Refinement
Of Cells

Improper Refinement
Of Cells

Figure 3.1: Improper Refinement

shown in Figure 3.1 will not be allowed. Second, we will require that the refined grids be *properly nested*. For any rectangular patch on level ℓ, the boundary may be:

1. a physical boundary where $\partial\Omega^\ell$ coincides with the physical boundary $\partial\Omega$,
2. a shared boundary with another grid at level ℓ (referred to as a *fine-fine* interface),
3. a coarse-fine interface with the next-finest level $\ell - 1$,
4. a mixture of these.

In particular, we will not allow $\partial\Omega^\ell$ to touch a valid level ℓ' cell for $\ell' < \ell - 1$; it may only see the physical boundary, other grids at this level, or the next-coarsest refinement level. See Figure 3.2.

Cells which have been tagged for refinement will be grouped together using the clustering algorithm of Berger and Rigoutsos [20] (see Section 5.4.1) to form efficient block-structured grids which cover the "tagged" regions. Grid efficiency is defined as the percentage of cells which are refined which were actually tagged for refinement.

$$\eta_{grid} = \frac{\text{Number of tagged cells}}{\text{Number of cells actually refined}} \tag{3.4}$$

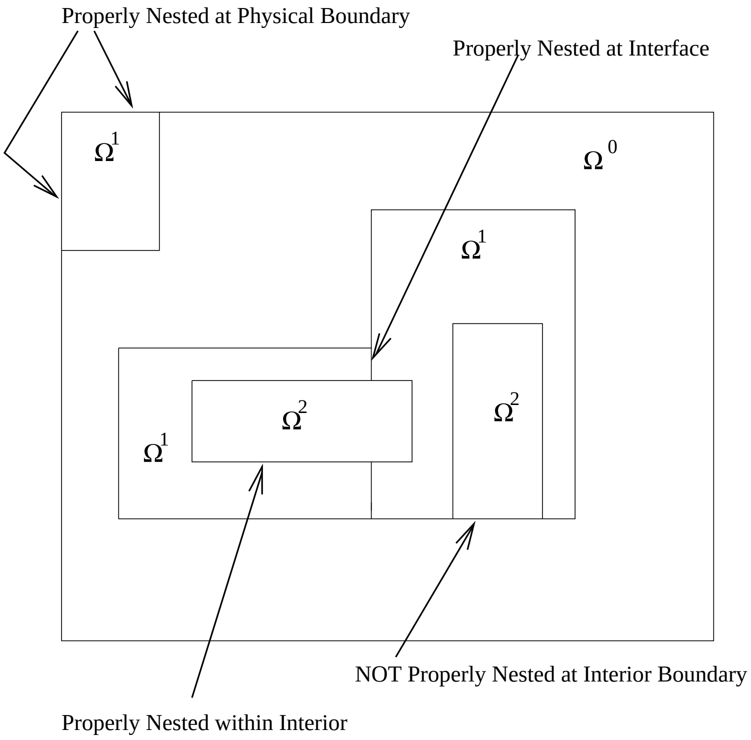

Properly Nested at Physical Boundary
Properly Nested at Interface
Ω^1
Ω^0
Ω^1
Ω^2
Ω^2
Ω^1
NOT Properly Nested at Interior Boundary
Properly Nested within Interior

Figure 3.2: Illustration of the proper nesting requirement

For simplicity, all level ℓ cells will share a global index space similar to that of level 0. For level 1 (the first level of refinement) this will be indexed $(0..(n^0_{ref}n^0_x)-1, 0..(n^0_{ref}n^0_y)-1)$. This will simplify communication of information between the coarse and fine levels. Conversion between the level 0 and level 1 indices is straightforward. A coarse cell (i^0, j^0) will correspond to the fine cells

$$(i^1, j^1) = ((n^0_{ref}i^0) + k, (n^0_{ref}j^0) + l) \quad \text{for } 0 \leq k, l \leq (n^0_{ref} - 1).$$

Conversely, a fine cell (i^1, j^1) is contained by the coarse cell

$$(i^0, j^0) = (\frac{i^1}{n^0_{ref}}, \frac{j^1}{n^0_{ref}}),$$

where integer division with rounding down is used.

3.1.2 Composite Operators and Level Operators

Since we have defined composite and level variables, we expect that we will need to define corresponding composite and level operators, which act on these variables.

In general, a *composite operator* will act on the composite solution on the multilevel hierarchy of grids. It will only compute values of the operator in the valid regions of a level. Definition of a composite operator will generally include special discretizations at coarse-fine interfaces to deal with the discontinuity in grid spacing in a reasonable way. On the other hand, a *level operator* will act on level variables, and as such will be essentially a single-grid operator which does not need to know about local refinements. It will be defined for all cells on a level, whether they are valid or covered by refinement. Note, however, that if the level operator is being applied to a refined level ($\ell > 0$), that it may need boundary conditions from the next coarser level. In general, our approach will be to enforce boundary conditions with coarser levels through the use of ghost cells (Section 2.1.1). If ghost cells around the grids on a refined level are filled with appropriate values prior to the application of the operator, then the level operator can be defined without knowledge of the level's position in the hierarchy of levels. For instance, in many cases we will want to use interpolated coarse-level solution values as boundary conditions for fine level operators. By first filling fine-level ghost cells with interpolated data, then applying the level operator, we can separate the implementation of the operator from the details of the AMR implementation.

In general, implementation of the level gradient, divergence, and Laplacian operators will be the same as those defined in Chapter 2, with the addition of coarse-fine boundary conditions from coarser levels. We will discuss only the simple gradient, divergence, and constant-coefficient Laplacian operators; the extension to more complicated operators is generally straightforward (see, for example, Bettencourt [22] or Propp [51].)

To define composite operators, we will extend the definitions of the gradient, divergence, and Laplacian operators from the edge-centered discretizations described in Section 2.5.1.

For Poisson's problem, we are solving

$$L\phi = \nabla \cdot \nabla \phi = \rho. \tag{3.5}$$

So, we will need to define a composite Laplacian operator. To simplify this, we will define the Laplacian as the divergence of the gradient, and then develop appropriate composite divergence and gradient operators which can then be incorporated into the definition of the Laplacian operator.

Gradient and Coarse-Fine Interpolation

To define a composite gradient operator, we will extend the edge-centered gradient defined in (2.52) to the case of a multilevel hierarchy of grids. The composite gradient will be defined on the valid edges of a level ℓ, $\Omega^{\ell,*}_{valid}$. Once we have defined the composite gradient, we will then define the level-operator gradient as a simple extension of the composite gradient operator.

On edges which are not coarse-fine interfaces, definition of the gradient is straightforward:

$$\begin{aligned} G^{comp}(\phi)^x_{i+\frac{1}{2},j} &= \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \\ G^{comp}(\phi)^y_{i,j+\frac{1}{2}} &= \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \end{aligned} \tag{3.6}$$

For computation of $G\phi$ at a coarse-fine interface, we will interpolate values for ϕ using both coarse and fine values. As an example, Figure 3.3 shows a coarse-fine interface with the coarse cells to the right of the interface and the fine cells to the left. To compute the x-component of the gradient across the interface, we will first interpolate a value into the ghost cell of the fine grid (the circled X's in Figure 3.3), and then use this interpolated value to compute a gradient:

$$\begin{aligned} G_{top} &= \frac{\phi^{I,top}_{i+1,j_{top}} - \phi_{i,j_{top}}}{\Delta x} \\ G_{bottom} &= \frac{\phi^{I,bot}_{i+1,j_{bot}} - \phi_{i,j_{bot}}}{\Delta x} \end{aligned} \tag{3.7}$$

To compute ϕ^I, we first use quadratic interpolation parallel to the coarse-fine interface using nearby coarse cells (marked as open circles in Figure 3.3) to get the intermediate points (marked with solid circles in Figure 3.3). Using this intermediate value along with two fine grid cells (marked with X's in Figure 3.3), another quadratic interpolation is used normal to the interface to get the appropriate ghost cell value (shown as circled X's in the figure).

We will henceforth denote this coarse-fine interpolation operator as $I(\phi^{fine}, \phi^{crse})$:

$$\phi^\ell = I(\phi^\ell, \phi^{\ell-1}) \text{ on } \partial\Omega^\ell \tag{3.8}$$

will mean that the ghost cell values for ϕ on level ℓ along the coarse-fine interface with level $\ell - 1$ are computed using this type of coarse-fine interpolation.

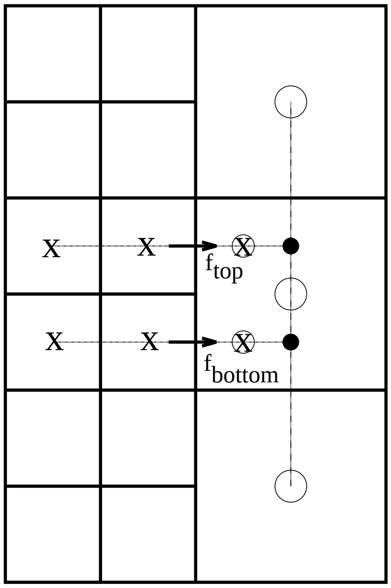


Figure 3.3: Interpolation at a coarse-fine interface

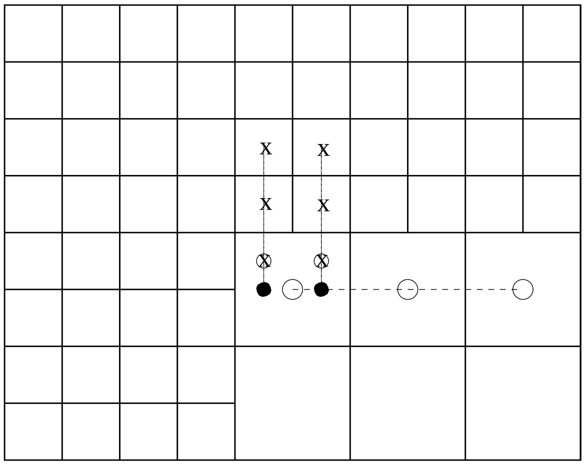


Figure 3.4: Modified interpolation stencil: Since the left coarse cell is covered by a fine grid, use shifted coarse grid stencil (open circles) to get intermediate values (solid circles), then perform final interpolation as before to get "ghost cell" values (circled X's). Note that to perform interpolation for the vertical coarse/fine interface, we will need to shift the coarse stencil down.

Since we will want to use this type of quadratic interpolation wherever possible to link the coarse- and fine-grid solutions, we must use different interpolation stencils for special cases like fine grid corners (Figure 3.4). If one of the coarse grid cells in the usual stencil is covered by a finer grid, we then shift the stencil so that only coarse cells in $(\Omega^c - P(\Omega^f))$ are used in the interpolation parallel to the coarse-fine interface. If a suitable coarse grid stencil does not exist, we then drop the order of interpolation and use whatever coarse cells we do have.

Definition of the level-operator gradient G^ℓ is straightforward. We will simply extend the definition of G^{comp}, which is only defined on the valid edges on a level $\Omega^{*,\ell}_{valid}$, to all edges on the level, $\Omega^{\ell,*}$. Specifically, away from coarse-fine interfaces with the coarser level $\ell - 1$, the composite operator will use the same stencil as the edge-centered gradient described in (2.52). At the coarse-fine interface with level $\ell - 1$, we will use the same coarse-fine boundary condition as was used for the composite gradient. The coarse-fine interpolation operator $I(\phi^\ell, \phi^{\ell-1})$ is used to compute ghost-cell values, which we can then use in the usual edge-centered gradient stencil.

Divergence and Reflux Divergence

We will also need composite and level divergence operators. We will define the composite divergence operator as a multilevel analog to the edge-centered divergence of (2.51), which is a cell-centered divergence of edge-centered fluxes.

For a cell in which none of the four edges are coarse-fine interfaces, this reduces to the normal edge-centered D operator:

$$(D^{comp}\mathbf{F})_{i,j} = \frac{F^x_{i+\frac{1}{2},j} - F^x_{i-\frac{1}{2},j}}{\Delta x} + \frac{F^y_{i,j+\frac{1}{2}} - F^y_{i,j-\frac{1}{2}}}{\Delta y}. \tag{3.9}$$

Note that for a cell in which none of the four edges are coarse-fine interfaces, this implies that the Laplacian operator (which is the composite divergence applied to the composite gradient) will reduce to the normal five-point Laplacian operator, which we would expect.

On the fine side of the coarse-fine interface, we assume that we have already computed an edge-centered flux on the coarse-fine interface. For instance, in the case of the Laplacian operator, we have defined gradients on the coarse-fine interfaces using the quadratic interpolation I to define boundary conditions with the coarse level. So, we can use (3.9) to compute the composite divergence for these cells as well. On the fine side of the coarse-fine interface, this will imply that the composite Laplacian once again reduces to the normal five-point Laplacian operator, using the interpolated ghost cell values ϕ^I. This also implies that the level-operator divergence D^ℓ, which will have no knowledge of any finer levels, will simply be the edge-centered divergence defined in (2.51) applied to all cells and edges (valid or covered) in Ω^ℓ.

For cells on the coarse side of a coarse-fine interface, we will replace the coarse-grid flux on the coarse-fine interface with the arithmetic average of the fine-grid fluxes. In the case of the coarse-grid cell in Figure 3.3, the divergence operator will be:

$$(D^{comp}\mathbf{F})_{i,j} = \frac{F^x_{i+\frac{1}{2},j} - \langle F^{x,fine}\rangle_{i-\frac{1}{2},j}}{\Delta x} + \frac{F^y_{i,j+\frac{1}{2}} - F^y_{i,j-\frac{1}{2}}}{\Delta y}, \tag{3.10}$$

where $\langle F^{x,fine}\rangle_{i-\frac{1}{2},j}$ is the arithmetic average of the fluxes on the fine-grid edges which cover coarse edge $(i-\frac{1}{2},j)$ (which is part of the coarse-fine interface with the fine level).

Assume that the coarse-grid fluxes $\mathbf{F}^{crse}$ can be extended to all edges in $\Omega^{\ell,*}$, including those covered by the coarse-fine interface edge between Ω^ℓ and $\Omega^{\ell+1}$. Adding and subtracting $\frac{F^{x,crse}_{i-\frac{1}{2},j}}{\Delta x}$ to the right hand side of (3.10), we get:

$$\begin{aligned}(D^{comp}\mathbf{F})_{i,j} &= \frac{F^x_{i+\frac{1}{2},j} - F^{x,crse}_{i-\frac{1}{2},j}}{\Delta x} - \frac{\langle F^{x,fine}\rangle_{i-\frac{1}{2},j} - F^{x,crse}_{i-\frac{1}{2},j}}{\Delta x} + \frac{F^y_{i,j+\frac{1}{2}} - F^y_{i,j-\frac{1}{2}}}{\Delta y} \\ &= (D^{crse}\mathbf{F}^{crse})_{i,j} - \frac{1}{\Delta x}\left(\langle F^{x,fine}\rangle_{i-\frac{1}{2},j} - F^{x,crse}_{i-\frac{1}{2},j}\right), \end{aligned} \tag{3.11}$$

where D^{crse} is the coarse-level edge-centered divergence operator. By doing this, we have split the composite divergence on the coarse side of the coarse-fine interface into the coarse-level operator plus a correction for the effect of the fine grid. This will prove to be very useful in our implementation, so we will define some associated notation.

As in (3.11), it will often be necessary to compute the difference between coarse and averaged fine edge-centered values on coarse-fine interfaces. To do this efficiently, we define a *flux register* $\delta F^{\ell+1}$, which will store the difference in the edge-centered quantity F on the coarse-fine interface between level ℓ and $\ell+1$. $\delta F^{\ell+1}$ will be owned by the fine level $\ell+1$ because it represents information on the boundary of level $\ell+1$. However, it will also have coarse-level (ℓ) grid spacing and indexing because it will generally be used to correct coarse-grid values with the appropriately averaged fine-grid values. See Figure 3.5. Note that the sign of the contributions to δF is such that the flux register represents the amount which must be added to the coarse grid fluxes to ensure agreement with the fine grid fluxes.

If we define the *reflux divergence* D_R as the coarse-level edge-centered divergence applied to edge-centered fluxes on the coarse-fine interface, then we can re-write (3.11) as

$$(D^{comp}F)_{i,j} = (D^{\ell}\mathbf{F}^{\ell})_{i,j} + D_R(\delta F^{\ell+1})_{i,j}, \tag{3.12}$$

where

$$\delta F^{\ell+1} = -F^{\ell} + \langle F^{\ell+1}\rangle \quad \text{on} \quad \partial\Omega^{\ell+1}. \tag{3.13}$$

For a coarse cell to the right of a coarse-fine interface (as in Figure 3.3),

$$D_R(\delta F^{\ell+1})_{ij} = -\frac{1}{\Delta x}(\delta F^{\ell+1})_{i-\frac{1}{2},j}. \tag{3.14}$$

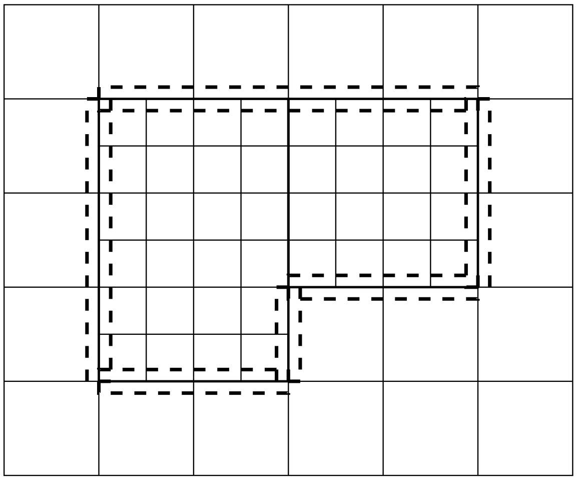


Figure 3.5: Flux register along $\partial\Omega^1$: the dashed lines represent the edge-centered flux register defined along the coarse-fine interface. Note that the flux register has coarse-grid spacing.

For a coarse cell to the left of a coarse-fine interface,

$$D_R(\delta F^{\ell+1})_{ij} = \frac{1}{\Delta x}(\delta F^{\ell+1})_{i+\frac{1}{2},j}. \tag{3.15}$$

The y-direction is similar. Note that D_R only affects the set of coarse cells immediately adjacent to the coarse-fine interface.

This will prove to be a very useful tool, in that we have separated the composite operator into the application of a single-level operator D^{crse} and a correction for the effect of finer levels. There is no reason why the reflux-divergence correction piece cannot be applied separately from the single-level piece. In fact, in many situations in time-dependent algorithms, this separation of coarse operator and fine-level correction will become necessary.

3.2 Solving Poisson's Equation on a Multilevel Hierarchy

In this section, we will describe our approach to solving Poisson's equation on a multilevel hierarchy of grids. We will first describe the discretization of our composite Laplacian operator, including a motivation of why we take so much care developing composite operators. Then, we describe our multilevel solution algorithm, and present an example demonstrating the performance of the algorithm.

3.2.1 Composite Laplacian – Elliptic Matching

First, we can define the level-operator Laplacian, which will be the level-operator divergence D^ℓ applied to the level-operator gradient G^ℓ:

$$L^\ell = D^\ell G^\ell. \tag{3.16}$$

Recall that the level-operator gradient uses the coarse-fine interpolation operator I to compute boundary conditions with a coarser level $\ell - 1$. This means that L^ℓ will be the single-grid Laplacian operator L defined in (2.9), with the addition of the coarse-fine interpolation operator I to provide boundary conditions with level $\ell - 1$ where necessary:

$$\begin{aligned} (L^\ell \phi)_{i,j} &= \frac{(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j})}{h^2} \text{ on } \Omega^\ell \\ \phi^\ell &= I(\phi^\ell, \phi^{\ell-1}) \text{ on } \partial\Omega^\ell. \end{aligned} \tag{3.17}$$

As mentioned earlier, we will define the composite Laplacian as the composite divergence applied to the composite gradient:

$$L^{comp}\phi = D^{comp}G^{comp}\phi \tag{3.18}$$

On the coarse grid away from the refined patches the composite operator looks the same as the single-grid Laplacian operator from (2.9):

$$L^{crse}(\phi)_{i,j} = \frac{\phi^c_{i+1,j} + \phi^c_{i-1,j} + \phi^c_{i,j+1} + \phi^c_{i,j-1} - 4\phi^c_{i,j}}{h_c^2}. \tag{3.19}$$

Likewise, on the fine grid away from coarse-fine interfaces the composite operator looks like the fine-grid version of the single-grid Laplacian operator:

$$L^{fine}(\phi)_{i,j} = \frac{\phi^f_{i+1,j} + \phi^f_{i-1,j} + \phi^f_{i,j+1} + \phi^f_{i,j-1} - 4\phi^f_{i,j}}{h_f^2}. \tag{3.20}$$

To define the composite operator where the normal stencils of the coarse and fine operators cross a coarse-fine interface, we first break the Laplacian into a flux-differencing formulation using a control volume around each cell. We can then write the Laplacian as the cell-centered divergence of edge-centered fluxes:

$$\begin{aligned} L(\phi)_{i,j} &= \nabla \cdot \mathbf{F} \qquad (3.21) \\ &= \frac{F^x_{i+\frac{1}{2},j} - F^x_{i-\frac{1}{2},j}}{\Delta x} + \frac{F^y_{i,j+\frac{1}{2}} - F^y_{i,j-\frac{1}{2}}}{\Delta y} \end{aligned}$$

where

$$\mathbf{F} = \nabla\phi. \tag{3.22}$$

Note that the fluxes are edge-centered quantities; at a coarse-fine interface, they will be defined on the interface. For the operator on the coarse side of the interface, the coarse flux will be the average of the fluxes used by the fine operator. Using edge-centered fluxes at the coarse-fine interface greatly simplifies the construction of the Laplacian operator across coarse-fine interfaces.

Recall that the composite gradient operator G^{comp} on a coarse-fine interface is defined through the use of the quadratic interpolation operator I to link coarse and fine levels, and the composite divergence operator links coarse and fine levels by using the averaged fine-grid fluxes on the coarse-fine interface to define the divergence on the coarse side of the interface.

Applying the flux register and reflux-divergence notation to the definition of the composite Laplacian operator, the complete description of the composite Laplacian L^{comp} on the valid region cells of level ℓ will be:

$$\begin{aligned} L^{comp}\phi^{comp}_{i,j} &= L^{\ell}\phi^{\ell}_{i,j} + D_R(\delta F^{\ell+1})_{i,j} \\ \phi^{\ell} &= I(\phi^{\ell}, \phi^{\ell-1}) \text{ on } \partial\Omega^{\ell} \\ \delta F^{\ell+1} &= \langle G^{\ell+1}\phi^{\ell+1}\rangle - G^{\ell}\phi^{\ell} \\ \phi^{\ell+1} &= I(\phi^{\ell+1}, \phi^{\ell}) \text{ on } \partial\Omega^{\ell+1} \end{aligned} \tag{3.23}$$

In words, the Laplacian is the single-level operator L^{ℓ} plus a reflux-divergence correction to account for the effect of a finer level (if one exists). Boundary conditions for the Laplacian operator between this level and a coarser level $\ell - 1$ (if one exists) are enforced by using the quadratic interpolation operator I to fill ghost cells around Ω^{ℓ}. The correction for the effects of a finer level is performed through a reflux-divergence of the difference between the coarse and fine fluxes along the coarse-fine interface between levels ℓ and $\ell + 1$. For the Laplacian operator, the flux is defined as the gradient of ϕ. To compute the fine-level $(\ell + 1)$ gradient of ϕ needed for the reflux-divergence correction, we fill ghost cells around the finer $\ell + 1$ level using quadratic interpolation between the level ℓ and level $\ell + 1$ solutions.

In short, for the operators defined in the this section, the basic philosophy will be to always compute boundary conditions with coarser levels using quadratic coarse-fine interpolation, while enforcing flux-matching with finer levels using reflux-divergences of the difference in the coarse and fine fluxes.

Elliptic Matching

We have taken quite a bit of care while defining the composite operators we will use. In this section, we will present a case explaining why we use such complicated operators.

When solving Poisson's equation on a multilevel hierarchy, care must be taken to ensure that the appropriate smoothness in the solution is maintained across the coarse-fine interface. Since solutions to elliptic equations like Poisson's equation are nonlocal in nature, we expect that a lack of smoothness at the coarse-fine interface will affect the solution in a global way.

The simplest approach would be to solve Poisson's equation on the coarse grid, where the source term on the coarse grid, ρ^c, is the average of ρ^f (the source term defined on the fine grid) where the coarse grid is covered by the fine grid. Then we could solve the problem on the refined domain, using interpolated values from the coarse solution as boundary conditions for the fine level. Unfortunately, it has been shown ([5]) that the resulting composite solution contains an error which scales with the coarse grid spacing. In other words, we are not attaining the increased accuracy we would expect from a calculation on a refined mesh.

The problem with this scheme is that the coarse and fine solutions are not sufficiently linked. Information is passed from the coarse grid to the fine grid in the form of a Dirichlet boundary condition, but the coarse solution is not modified by the fine solution in any way. This lack of communication of information from the fine solution back to the coarse solution causes a discontinuity in $\frac{\partial \phi}{\partial n}$ which is $O(h_c)$. Since the derivative of a discontinuous first derivative of the solution will look like a $\delta-$function in the second derivative, our solution looks like:

$$L\phi + C\delta(x_{cf})(\frac{\partial \phi}{\partial n}^{fine} - \frac{\partial \phi}{\partial n}^{crse}) = \rho \tag{3.24}$$

In effect, we have created a singular charge on the coarse-fine interface which is corrupting the solution. This charge is proportional to the mismatch in the derivatives of the coarse and fine solutions, and is $O(h_c)$. Bai and Brandt [10] note that for a similar approach, the coarse solution away from a singular source distribution is degraded because of a lack of conservation of source strength between the problems being solved on the coarse grid and on the fine grid.

In order to attain the desired fine-grid accuracy for solutions to Poisson's equation and to avoid corruption of the solution by coarse-fine interface error, we will need to ensure that the composite solution satisfies both Dirichlet *and* Neumann matching conditions at the coarse-fine interfaces. This is the *elliptic matching* condition.

Essentially, the problem with this algorithm is that it did not use the composite operators described in Section 3.1.2. Interpolating using both coarse and fine grids and using the same fluxes for both coarse and fine grids links the two solutions enough to satisfy the elliptic matching condition and fix the coarse-fine interface problem. We can then attain the improved accuracy expected from refinement.

3.2.2 Truncation Error Analysis

Quadratic interpolation is the minimum necessary to maintain second-order accuracy globally. We will use the gradient operator in the construction of the Laplacian, which is a second-derivative operator; it involves a division by h^2. If an interpolated quantity has a truncation error of $O(h^p)$, division by h^2 in the second derivative results in a truncation error of $O^{(p-2)}$. If we define $\phi^e_{i,j}$ as the exact composite solution,

$$\phi^{e,\ell}_{i,j} = \varphi(x_i, y_j) \quad \text{on } \Omega^\ell_{valid}, \tag{3.25}$$

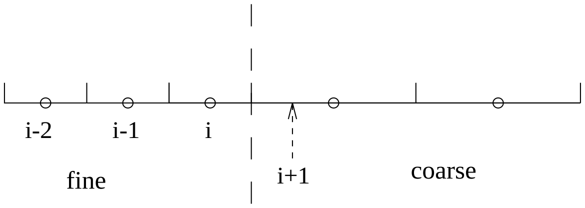


Figure 3.6: Sample one-dimensional coarse-fine interface

then the truncation error τ is defined on the valid regions of the level ℓ grids as:

$$\tau_{i,j} = \rho_{i,j} - L^{comp}(\phi^e)_{i,j}. \tag{3.26}$$

For the one-dimensional example shown in Figure 3.6, where ϕ^I_{i+1} is the interpolated value, and $\phi_{i+1} - \phi^I_{i+1} = O(h^p)$, we get:

$$\begin{aligned} \frac{\partial^2 \phi}{\partial x^2}|_i &= \frac{\phi_{i+1} + \phi_{i-1} - 2\phi_i}{h^2} + O(h^2) \\ &= \frac{\phi^I_{i+1} + \phi_{i-1} - 2\phi_i}{h^2} + \frac{\phi_{i+1} - \phi^I_{i+1}}{h^2} + O(h^2) \end{aligned} \tag{3.27}$$

so the error is $O(max(h^{p-2}, h^2))$. Even with quadratic interpolation ($p = 3$), there is still an $O(h)$ error at the coarse/fine interface.

Since the discretization of the Laplacian on the interiors of grids away from coarse fine interfaces is $O(h^2)$, we lose one order of accuracy along the coarse-fine interface due to the coarse-fine interpolation error (along with the division by h^2 in the Laplacian operator). The question then arises, "Does this coarse-fine error degrade the accuracy of the global solution?" Since the coarse/fine interface is a set of codimension one, we have observed that we can lose one order of accuracy and still be $O(h^2)$ globally, similar to what we have observed at the physical boundary (see Section 2.2.1). This cannot be improved by using higher-order interpolation; while the gradient at Ω^ℓ would be more accurate, the truncation error in the first coarse cell would still be $O(h)$, due to the lack of cancellation in the error in the gradient on the two edges.

Using a modified equation analysis, Johansen [43] has demonstrated that, in fact, the contribution of the higher truncation error at coarse-fine interfaces is indeed $O(h^2)$. In essence, this is the same reasoning presented in Section 2.2.1, repeated here for completeness. While this is not a rigorous analysis, it does provide some insight, and agrees with what we see in practice. First, we define the truncation error, τ, as

$$\tau_{i,j} = \overline{\rho} - L(\phi^e)_{i,j} \tag{3.28}$$

where $\overline{\rho}$ is the discrete approximation to ρ used in the numerical method and $L(\phi^e)_{i,j}$ is the discrete operator applied to the exact solution ϕ^e. Then, as we have seen, we have these estimates for $\tau_{i,j}$:

$$\tau_{i,j} = \begin{cases} O(h^2) & \text{for interior cells} \\ O(h) & \text{for cells adjacent to a C/F interface} \end{cases} \tag{3.29}$$

If we define the solution error $\xi_{i,j} = \phi_{i,j} - \phi^e_{i,j}$, then the error satisfies the error equation

$$L\xi = \tau \tag{3.30}$$

The expectation is that the contribution of each cell to ξ is proportional to the total charge on that cell. For an interior cell, this is $\tau_{i,j} \times h^2 = O(h^4)$; for a boundary cell, it is $\tau_{i,j} \times h^2 = O(h^3)$. There are $O(\frac{1}{h^2})$ interior cells, for a total contribution of $O(h^2)$ to ξ, while there are only $O(\frac{1}{h})$ boundary cells, resulting in a total contribution of $O(h^2)$ as well.

3.2.3 Multilevel Multigrid Iteration Algorithm

The algorithm described here is the logical extension of the multigrid algorithm described in Section 2.2.3 to a multilevel hierarchy of locally refined grids. Our algorithm is a variant of one first proposed by Brandt [24], and extended by Bai and Brandt [10]. Thompson and Ferziger [65] used a similar multigrid algorithm to compute steady incompressible flow, and Almgren, Buttke, and Colella [7] developed a node-centered version for use in a fast vortex method.

The algorithm described here is based on that in Martin and Cartwright [47], which is itself a cell-centered extension of the node-centered algorithm of Almgren, Buttke, and Colella. [7]. A similar algorithm has been used for steady compressible flow by Dudek [34], and for semiconductor device simulation by Bettencourt [22]. The only substantial modification in the algorithm from [47] is the addition of a conjugate-gradient solver for unions of rectangles for the coarsest level, instead of repeated relaxation, as was used in the previous work (see Section 3.2.5). This will allow elliptic solves which have a coarsest level $\ell_{base} > 0$, because we will generally have to solve on an arbitrary union of rectangles at the bottom of the multigrid V-cycle, instead of a single $1 \times n$ grid.

For simplicity, we will first describe the multilevel solution algorithm for the case where we are solving over the entire domain ($\ell_{base} = 0$) and $n_{ref} = 2$. Then we will extend the algorithm to cover more general cases.

We want to solve

$$L^{comp}(\phi) = \rho \quad \text{on} \quad \Omega^{\ell_{\mathrm{base}}} \tag{3.31}$$

where $L^{comp}(\phi)$ is the composite Laplacian operator described in Section 3.2.1.

For each refinement level from $\ell = 0$ to ℓ_{max}, we will obviously need to store Ω^ℓ, ϕ^ℓ, and ρ^ℓ, where ϕ^ℓ and ρ^ℓ are only defined on $\Omega^\ell - \mathrm{P}(\Omega^{\ell+1})$, that is, wherever Ω^ℓ is not covered by a finer grid. Since we are using the residual-correction formulation, for each level we will also have to define the residual R^ℓ and the correction e^ℓ on the entire Ω^ℓ (including the covered regions of Ω^ℓ).

In addition to the composite Laplacian L^{comp}, which is defined over the entire hierarchy of levels, and the level-operator Laplacian L^ℓ, we will also define the composite Laplacian on level ℓ,

$L^{comp,\ell}(\phi^\ell, \phi^{\ell+1}, \phi^{\ell-1})$, which is defined on the valid region of level ℓ:

$$L^{comp,\ell}(\phi^\ell, \phi^{\ell+1}, \phi^{\ell-1}) = L^{comp}\phi \quad \text{on} \quad \Omega^\ell - \mathrm{P}(\Omega^{\ell+1}) \tag{3.32}$$

Recall that away from the boundaries of Ω^ℓ_{valid}, $L^{comp,\ell}$ is simply the normal $L^{h_\ell}\phi^\ell$ that we are used to dealing with. In cells which abut the $\Omega^\ell/\Omega^{\ell-1}$ boundary, we interpolate values into the border cells using the quadratic interpolation operator I and then evaluate L^{h_ℓ} as usual. Finally, for cells adjacent to the $\Omega^\ell/\Omega^{\ell+1}$ boundary, we use our flux matching condition to generate the fluxes across the boundary. Thus, we always interpolate coarse grid information as mentioned earlier, and we always use the flux matching condition to represent the influence of the finer grids.

Note that we have explicitly shown the dependence of $L^{comp,\ell}$ on both the coarser-level solution (in the form of quadratic interpolation with $\phi^{\ell-1}$) and the finer-level solution (in the form of the flux-matching condition with $\phi^{\ell+1}$). In a similar way, we will explicitly show the dependence of L^ℓ on the coarser-level solution through the coarse-fine boundary condition $I(\phi^\ell, \phi^{\ell-1})$ by referring to the level-operator Laplacian as $L^\ell(\phi^\ell, \phi^{\ell-1})$.

We will also need an operator which performs a point relaxation for Poisson's equation. So, we define GSRB_LEVEL(e^ℓ, R^ℓ, h_ℓ) on Ω^ℓ. This performs one iteration of Gauss-Seidel with Red-Black ordering on the data on level ℓ. This operator has no information about other levels, although it should know the appropriate operators and boundary conditions to relax on each level. Therefore, this operator looks like:

$$e^\ell_{i,j} := e^\ell_{i,j} + \lambda\{L^\ell(e^\ell, e^{\ell-1} = 0) - R^\ell_{i,j}\} \tag{3.33}$$

with red-black ordering. As before, red-black ordering means that we relax using two passes through the domain in a checkerboard pattern: on the first pass, we relax on points where (i+j) is even (the RED pass); on the second, we relax on points where (i+j) is odd (the BLACK pass). Note that, because the GSRB_LEVEL is designed to be unaware of both coarser and finer levels, the relaxation uses L^ℓ with all the coarse grid information set to 0. In other words, we use the coarse-fine interpolation operator $I(\phi^\ell, 0^{\ell-1})$, where $0^{\ell-1}$ denotes a coarse level $\ell - 1$ grid with zeros in all the cells. For interior cells, we use the normal relaxation parameter $\lambda_{interior} = \frac{h^2}{4}$.

For each level, the residual will contain two components. First, as in normal multigrid relaxation, the residual on level ℓ contains the residual from higher (finer) levels, mostly the low wavenumber error that is not damped out by the GSRB iterations at the finer levels. In addition, there is the residual from the operators on level ℓ where there are no overlying finer grids. If there is no overlying fine grid, then we are starting our multigrid V-cycle on this level; otherwise, we are simply continuing the multigrid relaxation which was begun on the finer levels.

The multilevel multigrid algorithm we will employ is described in pseudocode form in Figure 3.7. The function AMRPoisson(ϵ) will call the recursive multigrid iteration function MGRelax(ℓ) until the maximum residual has been decreased by a factor of ϵ.

The algorithm is structured like the multigrid algorithm for a single grid, described in Section 2.2.3 – we start at the finest levels, then progressively coarsen and relax our way down the V-cycle, then solve on the coarsest level, then interpolate and relax our way back up the V-cycle. The difference is that in this case, the data to which we are applying our various operators may not be defined on the entire physical domain at that level. We will use the same interpolation and restriction operators that we used in the single-grid multigrid algorithm: $R_\ell^{\ell-1}$ will be simple arithmetic averaging, and $I_{\ell-1}^{\ell}$ will be piecewise constant interpolation.

Since we compute the coarser-level residual on the uncovered regions of the coarser grids

```
AMRPoisson($\epsilon$)
    $Res := \rho - L^{comp}(\phi)$.
    while ($|Res| > \epsilon|\rho|$)
        $Res := \rho - L^{comp}(\phi)$.
        MGRelax($\ell^{\max}$).
    end while
end AMRPoisson
```

```
MGRelax($\ell$):
    if ($\ell = \ell^{max}$) then $Res^\ell := \rho^\ell - L^\ell(\phi^\ell, \phi^{\ell-1})$
    if ($\ell > 0$) then
        $\phi^{\ell,save} := \phi^\ell$
        $e^{\ell-1} := 0$
        $e^\ell := \mathrm{GSRB\_LEVEL}(\mathrm{e}^\ell, \mathrm{Res}^\ell, \mathrm{h}_\ell)$
        $\phi^\ell := \phi^\ell + e^\ell$
        $Res^{\ell-1} := R_\ell^{\ell-1}(Res^\ell - L^\ell(e^\ell, e^{\ell-1}))$ on $\mathrm{P}(\Omega^\ell)$
        $Res^{\ell-1} := \rho^{\ell-1} - L^{comp,\ell-1}(\phi^{\ell-1}, \phi^\ell, \phi^{\ell-2})$ on $\Omega^{\ell-1} - \mathrm{P}(\Omega^\ell)$
        MGRelax($\ell - 1$)
        $e^\ell := e^\ell + I_{\ell-1}^\ell(e^{\ell-1})$
        $Res^\ell := Res^\ell - L^\ell(e^\ell, e^{\ell-1})$
        $\delta e^\ell := 0$
        $\delta e^\ell := \mathrm{GSRB\_LEVEL}(\delta \mathrm{e}^\ell, \mathrm{Res}^\ell, \mathrm{h}_\ell)$
        $e^\ell := e^\ell + \delta e^\ell$
        $\phi^\ell := \phi^{\ell,save} + e^\ell$
    else    solve/relax $L^0 e^0 = Res^0$ on $\Omega^0$
        $\phi^0 := \phi^0 + e^0$
    end if
end MGRelax
```

Figure 3.7: Pseudocode description of AMR Poisson multigrid algorithm

using the composite operator $L^{comp,\ell-1}(\phi^{\ell-1}, \phi^{\ell}, \phi^{\ell-2})$, note that we update ϕ^{ℓ} with the current correction before computing the coarser residual, in order to compute a residual which reflects corrections made on the finer level. However, on the way back up the multigrid V-cycle, we will want to add the correction to the original value for ϕ, which is why we save the original uncorrected value for ϕ^{ℓ} as $\phi^{\ell,save}$.

When we arrive at the coarsest level, we have one domain. We can then iterate on $L^0 e^0 = Res^0$ on Ω^0 using the single-grid multigrid algorithm described in Section 2.2.3. Once that is done, we update the coarse level solution, $\phi^0 := \phi^0 + e^0$, and start back up the V-cycle.

On the way up the multigrid V-cycle, we must modify the algorithm slightly. First, we update the fine grid (level ℓ) correction:

$$e^{\ell} = e^{\ell} + I^{\ell}_{\ell-1}(e^{\ell-1}). \tag{3.34}$$

However, now we cannot go directly to a GSRB_LEVEL iteration, because we now have a coarse grid correction which we will need to use as a boundary condition. We handle this the same way we handle any problem with inhomogeneous boundary conditions: we put the problem in residual-correction form to make the boundary conditions homogeneous. So, we first must modify the residual:

$$Res^{\ell} := Res^{\ell} - L^{\ell}(e^{\ell}, e^{\ell-1}). \tag{3.35}$$

We then define a correction to the correction, δe^{ℓ}, set it to 0, and then perform a GSRB_LEVEL operation on it:

$$\delta e^{\ell} := \mathrm{GSRB\_LEVEL}(\delta \mathrm{e}^{\ell}, \mathrm{Res}^{\ell}, \mathrm{h}_{\ell}). \tag{3.36}$$

Then, we can update the correction and the copy of ϕ^{ℓ} which we had saved:

$$e^{\ell} := e^{\ell} + \delta e^{\ell} \tag{3.37}$$

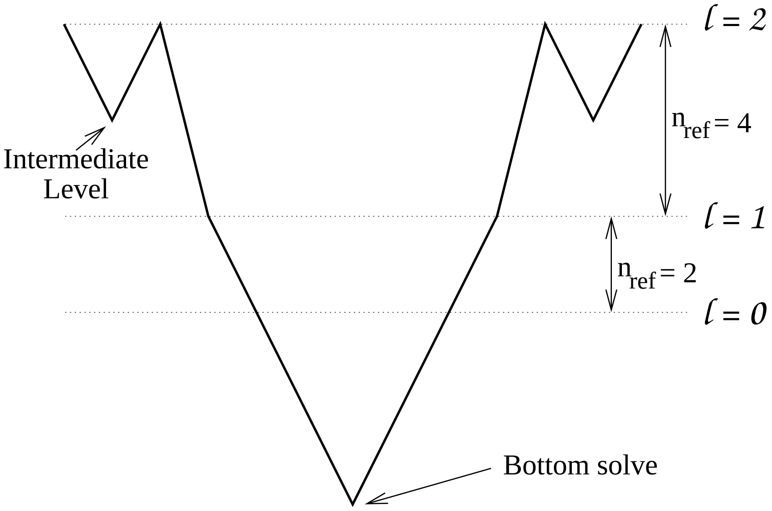


Figure 3.8: Multigrid with $n_{ref} \neq 2$. Because $n^1_{ref} = 4$, we perform an intermediate coarsening in the multigrid cycle before coarsening from level 2 to level 1.

$$\phi^\ell := \phi^{\ell,save} + e^\ell. \tag{3.38}$$

3.2.4 Extension to $n_{ref} = 2^p, p > 1$

When n_{ref} is two, the multigrid coarsening and injection is straightforward, since the coarsenings used in the multigrid algorithm correspond to existing levels of refinement. This is not the case when n_{ref} is greater than two, however; we still want to coarsen by a factor of two for multigrid, but this will result in intermediate multigrid levels which do not correspond to the data in our multilevel grid hierarchy.

In this case, we will modify the algorithm slightly by doing a mini-multigrid V-cycle, coarsening the fine grids by repeated factors of two until the next coarsening would result in the same grid spacings as an existing level of data in the AMR hierarchy. A schematic of this cycle is shown in Figure 3.8. In this example, $n^1_{ref} = 4$. So, we first relax on level 2, then coarsen the level 2 grids down to the intermediate level shown, coarsen the residual and correction to this intermediate level, and relax using GSRB-LEVEL. Then, we interpolate the correction back to level 2 and relax on level 2 again. Then, we coarsen the level 2 residual down to level 1 and continue on our way. Since n^0_{ref} is 2, we can relax on level 1 and then coarsen directly down to level 0, from which we once again coarsen as far as possible, solve, and then proceed back up the hierarchy. Once the solution has been relaxed on level 1, we interpolate directly to level 2, relax on level 2, then coarsen to the intermediate level again, where we relax again before interpolating the solution back up to level 2 and performing a final relaxation.

The reason why we relax on the intermediate levels and then interpolate back to the fine level before coarsening to the next coarsest AMR level is that coarse-fine boundary conditions are simplified. Since we are using the residual-correction form of the equation, the coarse-fine boundary conditions on the correction are a homogeneous version of the coarse-fine interpolation discussed in Section 3.1.2. In this interpolation, we use the same coarse grid used in the level 2/level 1 interpolation, but with zeroes in all the cells. It is important for consistency that we use the same coarse grid for all the intermediate coarsenings, so that the distance of the coarse-cell values from the coarse-fine interface remains constant as we coarsen the grids.

Note also that for $n_{ref} = 8$ there would be *two* intermediate levels, for $n_{ref} = 16$ there would be three intermediate levels, etc. (Although in practice we rarely use $n_{ref} > 4$).

3.2.5 Extension to $\ell_{base} > 0$

In various places we will want to solve a multilevel elliptic problem on levels $\ell \geq \ell_{base}$ where $\ell_{base} > 0$. In this case, we are solving on all levels finer than (and including) level ℓ_{base}, with appropriate coarse boundary condition values provided from level $\ell_{base} - 1$ if necessary. This solution algorithm is similar to that presented in Section 3.2.3 for the levels finer than ℓ_{base}. When we reach level ℓ_{base}, we perform a procedure similar to that used for level 0 in Section 3.2.3, coarsening the level ℓ_{base} grids as much as possible. In general, we will not be able to reach a $1 \times n$ grid through repeated coarsenings of level ℓ_{base} grids. In most cases, we will reach a point where further coarsenings are impossible without destroying the "footprint" of the grids (Figure 3.10). In other words, further coarsening will result in a set of coarsened grids which, when re-refined, will not be the same as the original grids:

$$\Omega^{\ell,coarsest} \neq refine(coarsen(\Omega^{\ell,coarsest})).$$

When we reach this level, we then solve the resulting coarsened residual-correction equation exactly (or as exactly as possible) before starting back up the hierarchy. At present, we use a conjugate gradient solver [12] as a bottom solver. Our implementation of a conjugate gradient solver on a union of rectangles follows that of Bettencourt [22], and is detailed in Figure 3.9. Note that computations are carried out using composite operators over the union of grids. Note also that the problem as defined with composite operators over a union of grids is no longer symmetric, so the conjugate gradient approach is not guaranteed to work. In our case, the problem is simple enough that we have not experienced any difficulties; however, Bettencourt [22] found it necessary to use a Biconjugate Gradient Stabilized (BiCGStab) method for problems with strongly varying coefficients.

Coarse-fine boundary conditions are enforced by using the coarse-fine interpolation described in Section 3.1.2 to compute ghost-cell values for ϕ when computing the residual on level ℓ_{base}. Then, homogeneous coarse-fine interpolation (again keeping the coarse grid data constant) is used for the coarsenings of level ℓ_{base}, in the same way as for the intermediate multigrid levels in Section 3.2.4.

```
BottomSolve($\phi, b$)
    $res^{(0)} = L(\phi) - b$
    $corr^{(0)} = 0$
    for $(i = 1, 2, ...)$
        Smooth($corr^{(i-1)}, res^{(i-1)}$)
        Smooth($corr^{(i-1)}, res^{(i-1)}$)
        if ( $i > 1$ and $\rho_{\mathrm{i-2}} == 0.0$ ) return
        $\rho_{i-1} = Dot(corr^{(i-1)}, res^{(i-1)})$
        if $(i == 1)$
            $p^{(1)} = corr^{(0)}$
        else
            $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
            $p^{(i)} = corr^{(i-1)} + \beta_{i-1} * p^{(i-1)}$
        end if
        $q^{(i)} = L(p^{(i)})$
        $\alpha_i = \rho_{i-1}/Dot(q, p^{(i)})$
        $corr^{(i)} = corr^{(i-1)} + \alpha_i p^{(i)}$
        $res^{(i)} = res^{(i-1)} - \alpha_i q^{(i)}$
        $u^{(i)} = L(corr^{(i)}) - res^{(0)}$
        if ($\|u^{(i)}\| < tol * \|res^{(0)}\|$) return
    end for
    $\phi = \phi + corr^i$
end BottomSolve
```

Figure 3.9: Pseudocode for the conjugate gradient bottom solver

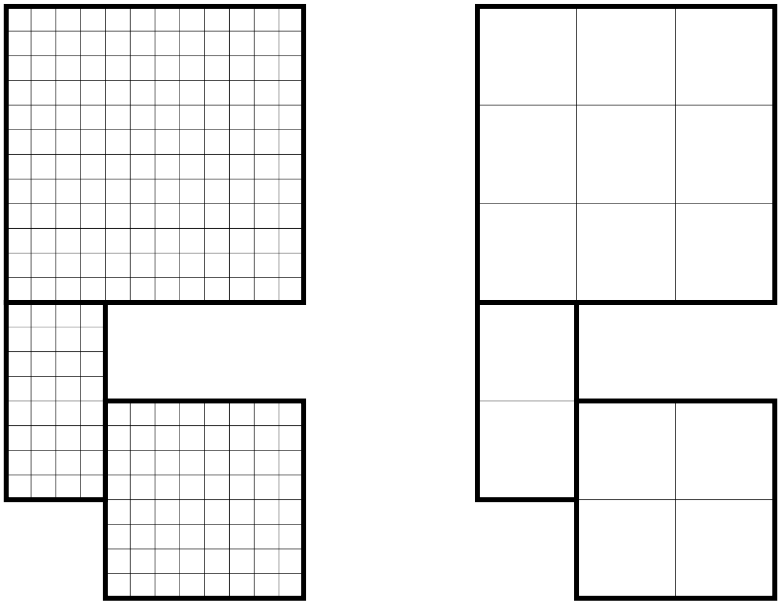


Figure 3.10: Best Coarsening: Grid configuration at right is the best possible coarsening of the grids at left.

3.2.6 Level Solves

In the AMR algorithm for the incompressible Euler equations which is described in the next chapter, we will also at times need to solve the elliptic equation on one level ℓ without solving on either finer or coarser grids. In this case, the algorithm will be the same as that of the algorithm in Section 3.2.5 if ℓ_{base} is also the finest level. We simply compute the residual on Ω^{ℓ} without taking the effect of finer levels (even if they do exist) into account. Then, we implement multigrid in the same way as in Section 3.2.5 for level ℓ_{base}: coarsen as far as possible, apply the conjugate-gradient bottom solver, and then refine back up to level ℓ.

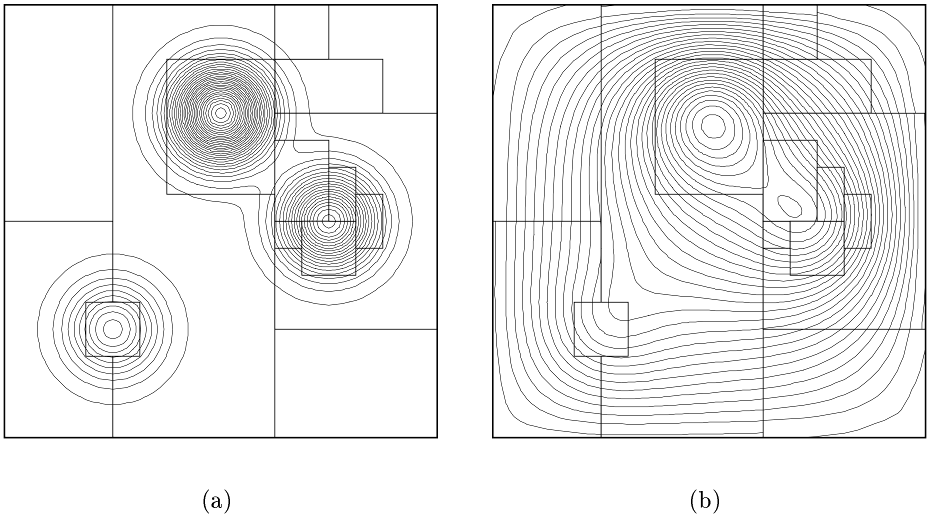

(a) (b)

Figure 3.11: AMR Poisson test problem (a) Source distribution, and (b) Solution

3.2.7 Performance of the Algorithm

The AMRPoisson code was tested on a sample problem with ρ equal to three Gaussian charges, as shown in Figure 3.11. To give an idea of grid placement, the grids used for a solution with two levels of refinement are shown as well. To judge the effects of adaptivity, we solved this problem with a series of coarser base grids, but with the same error tolerance. By doing this, we solve the problem to the same level of accuracy each time, but more levels of refinement become necessary as the base grid becomes coarser. The Richardson extrapolation error estimation algorithm of Section 5.3 was used to estimate the local truncation error of the solution; cells with estimated errors higher than the specified error tolerance were tagged for refinement. Cells marked for refinement were then clustered into unions of rectangles using the clustering algorithm of Berger and Rigoutsos [20], described in Section 5.4.1. By setting the error tolerance ϵ_{error} for the Richardson extrapolation

Base Grid Size	h = 1/64	1/128	1/256	1/512	1/1024	total
1024 × 1024	—-	—-	—-	—-	1048576	1048576
512 × 512	—-	—-	—-	262144	2304	264448
256 × 256	—-	—-	65536	10496	2304	78336
128 × 128	—-	16384	17280	10496	2304	46464
64 × 64	4096	15360	17152	10496	2304	49408

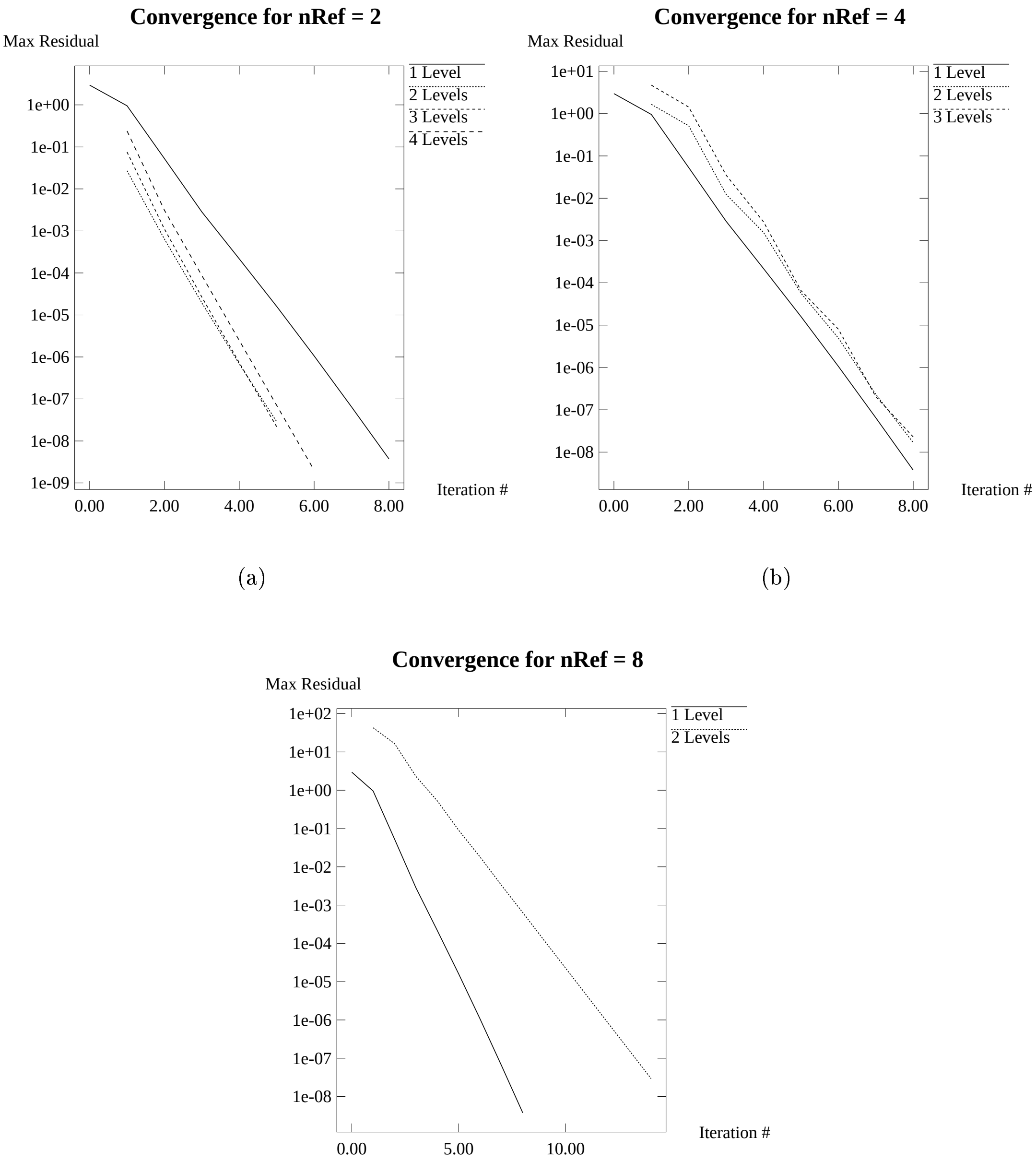
Table 3.1: Number of cells at each grid resolution, tabulated for different base grid sizes when solving sample problem

error tagging routine to be 0.0005, no refinements were needed for a base grid of 1024×1024, while one level of refinement was needed for a 512×512 base grid (and two levels were needed for a 256×256 base grid, etc.). For all of the solutions, the maximum error on the finest grid as computed using (3.39) was 4.27×10^{-4}.

$$Error^{\ell} = \text{Average}(\mathrm{L}^{\ell}(\phi^{\ell}, \phi^{\ell-1})) - \mathrm{L}^{\ell-1}(\text{Average}(\phi^{\ell})). \tag{3.39}$$

To show the effects of adaptivity on the resulting grid hierarchy, the total number of cells on each level is tabulated in Table 3.1. It is worth noting that, in every solution where refinement is employed, the number of cells at the finest resolution is constant at 2304, while the number of cells at the second finest resolution is constant at 10496. This points to the effectiveness of Richardson extrapolation as a consistent indicator of the necessary resolution for attaining a given level of accuracy in the solution.

The convergence history of this algorithm is shown in Figure 3.12 for refinement ratios of 2, 4, and 8 Adding local refinement to the solution did affect the convergence rates of the multigrid cycle somewhat. The convergence results are shown in Table 3.2. With no refinement, the Max(residual) was reduced by an average factor of 16.0 per multigrid cycle. In other words, the maximum of the residual after one full multigrid V-cycle was, on average, $\frac{1}{16.0}$ times the maximum residual at the



(a)

(b)

(c)

Figure 3.12: Multigrid Convergence for (a) $n_{ref} = 2$, (b) $n_{ref} = 4$, and (c) $n_{ref} = 8$

Refinement Ratio	1 level (no refinement)	2 levels	3 levels	4 levels
2	16.00	31.87	37.23	35.26
4	16.00	20.58	25.25	—-
8	16.00	5.23	—-	—-

Table 3.2: Convergence rates (average factor by which max(residual) is reduced for each multigrid iteration), tabulated for different refinement ratios and number of levels of refinement

start of the V-cycle. When one level of refinement (with a refinement ratio of 2) was added to the solution, the convergence rate increased to an average factor 31.9 reduction per multigrid cycle. A second refinement led to increased convergence, with an average reduction of 37.23. When a fourth level was added, however, the convergence rate decreased to an average factor 35.26 reduction in the max(residual).

Refinement ratios greater than 2 do appear to slow convergence somewhat. With a refinement ratio of 4, one level of refinement converged with an average factor 20.58 reduction in the residual, while for two levels, the max(residual) was reduced by an average factor of 25.25 per multigrid iteration. Using a refinement ratio of 8 led to a markedly poorer performance, however. One level of factor 8 refinement only showed an average reduction in the max(residual) of a factor of 5.3 per multigrid cycle. For the rest of this section, all timing results use a refinement ratio of two. We believe that the slower convergence rates for refinement ratios greater than two are a result of the intermediate V-cycles necessary in these computations.

To easily judge the effects of adaptivity, timings were normalized by the timing for the unrefined 1024×1024 solution, which was 58.11 sec on an SGI Power Challenge. Also, the total number of cells for all levels in each solution (including the non-valid portions where grids are overlain by finer grids) was recorded and likewise normalized by the total number of cells for the

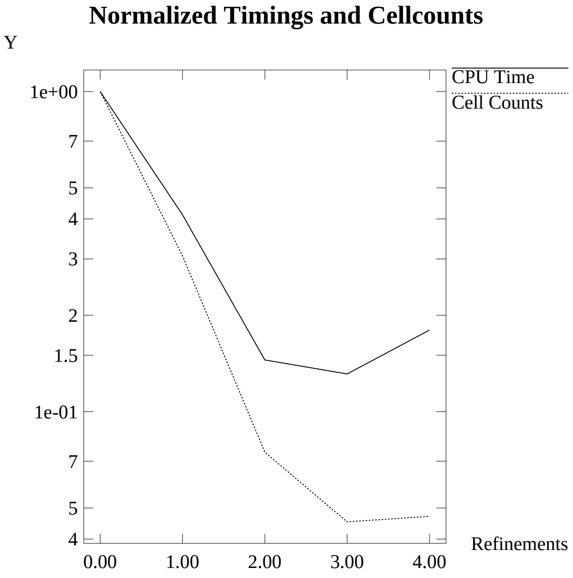


Figure 3.13: Normalized timings for the Poisson Solver

unrefined grid, 1048576. The total number of cells is an indicator of how much memory was used in the solution. A log plot of these normalized results appear in Figure 3.13. As can be seen, the total number of cells in the solution decreases with the number of refinement levels, ranging from 30.6% of the base number of cells with one level of refinement to 4.5% of the base number of cells with three levels of refinement. Adding a fourth level of refinement actually *increases* the total number of cells because so much of the base level is being refined (in this case, 93.75 of the domain is refined to level 1). The timings initially decrease strongly with additional levels of refinement, up to two levels of refinement, then level off at around 15% of the CPU time for the unrefined solution and actually rise slightly. This leveling off is due to the need when using Richardson extrapolation to compute a solution with $\ell - 1$ levels before generating a ℓth level. In other words, to compute a solution with two levels of refinement, first a single grid solution must be computed, then the error estimator defines the level 1 grids, which are then used to compute a two level solution, and then the error estimator is able to create the level 2 grids based on the error computed in the two level composite solution. Since the coarser levels are, in general, small in comparison to a finer base domain, this is generally inexpensive. However, with more levels of refinement, this can begin to offset the savings in computational time. In a sense, this is sub-optimal, because more effort is being spent on recomputing solutions on the coarse levels than on the finer levels, especially if the refined levels are small compared to the coarse levels. Bai and Brandt [10] suggest computing the initial solutions to less accuracy, increasing the solution accuracy as the number of levels increases. In their experience, this evens out the amount of work spent on coarser levels.

It should be noted, however, that once the break even point has been reached on CPU time, the additional refinement in this case still represents a savings in memory. Obviously, when the total number of cells increases, as is the case between three and four levels of refinement, CPU time will increase faster than the number of cells in the solution, due to the overhead of generating and managing the grid hierarchy.

3.3 Alternate Algorithm

In our time-dependent algorithm for the incompressible Euler equations, we will refine in time as well as space. Since different levels will be advanced using different timesteps, it will not generally be feasible to perform solves in the composite manner outlined in the previous sections. Following the example of Berger and Colella [18], we will structure our multilevel solution algorithm as a series of solves on individual levels, along with corrections to enforce the proper coarse-fine matching conditions. The level solves will consist entirely of operations on single levels (with no influence from finer levels) and interpolated boundary conditions from coarser grids if necessary. Once a solution based on level operators has been computed on all levels, we will need to correct to the solution to ensure that the composite solution satisfies the equations based on composite operators.

3.3.1 LevelSolve + Correction Formulation

In the case of Poisson's equation, this is straightforward. Using (3.23), we can re-cast the equation we are trying to solve on a given level ℓ as:

$$\begin{aligned} L^{comp,\ell}(\Phi^\ell) &= L^\ell(\Phi^\ell, \Phi^{\ell-1}) + D_R(\delta\Phi^{\ell+1}) = \rho^\ell \\ \Phi^\ell &= I(\Phi^\ell, \Phi^{\ell-1}) \ \text{ on } \ \partial\Omega^{\ell/\ell-1} \end{aligned} \tag{3.40}$$

Notice that we have explicitly included the coarse-fine interpolation operator, which represents the coarse-fine boundary condition on Φ with the coarser level $\ell - 1$. $\delta\Phi^{\ell+1}$ is the flux register which contains the mismatch in $\nabla\Phi$ along the $\ell/\ell+1$ interface, which is:

$$\delta\Phi^{\ell+1} = -G^\ell\Phi^\ell + < G^{\ell+1}\Phi^{\ell+1} > \quad \text{on} \ \ \partial\Omega^{\ell/\ell+1} \tag{3.41}$$

where $G^{\ell+1}\Phi^{\ell+1}$ is computed using the standard coarse-fine interpolation operator to compute ghost cell values for Φ^I.

This formulation leads to an obvious splitting into level operators and corrections. Let

$$\Phi = \phi + e \tag{3.42}$$

where ϕ^ℓ is the result of a level solve for ϕ and e is the correction field needed to ensure that Φ satisfies the composite equation. In this case, (3.40) becomes:

$$L\Phi \quad = \quad L^\ell\phi^\ell + D_R(\delta\Phi^{\ell+1}) + L^\ell e^\ell + D_R(\delta e^{\ell+1}) = \rho \tag{3.43}$$

$$\begin{aligned} \phi^{\ell} &= I(\phi^{\ell}, \phi^{\ell-1}) \text{ on } \partial\Omega^{\ell/\ell-1} \\ e^{\ell} &= I(e^{\ell}, e^{\ell-1}) \text{ on } \partial\Omega^{\ell/\ell-1} \end{aligned}$$

With a little rearranging, this becomes the level solve equation for ϕ:

$$\begin{aligned} L^{\ell}\phi^{\ell} &= \rho^{\ell} \qquad (3.44) \\ \phi^{\ell} &= I(\phi^{\ell}, \phi^{\ell-1}) \text{ on } \partial\Omega^{\ell} \end{aligned}$$

along with an associated equation for e:

$$\begin{aligned} L^{\ell}e^{\ell} &= -D_R(\delta\phi^{\ell+1}) - D_R(\delta e^{\ell+1}) \qquad (3.45) \\ e^{\ell} &= I(e^{\ell}, e^{\ell-1}) \text{ on } \partial\Omega^{\ell} \end{aligned}$$

which we can solve through iteration, also using level solves.

All that remains is to embed this formulation in an iterative algorithm. There are actually two different ways to do this. While the initial level solves for ϕ must be ordered from the coarsest level followed by the successively finer levels because of the coarse level boundary conditions, the correction need not be done that way. The correction may be solved from coarsest level to finest level (bottom-up iteration), or it may be solved from the finest level down to the coarsest level (top-down iteration). We will look at each in turn. Both algorithms were then tested using a similar test problem to that used in Section 3.2.7, but with only one Gaussian source in the center (to make visualization simpler).

3.3.2 Bottom-Up Iteration

In this algorithm, we first do a series of single level solves for ϕ, solving from the coarsest level up to the finest, using the coarser level solution as a boundary condition for the current level.

Then, starting at the coarsest level, we solve for e, which is the correction due to the effect of the finer level solution. We then iterate on the correction until the composite residual is sufficiently reduced.

The source for the correction on each level ℓ has two components: the mismatch between the current level and the finer $(\ell+1)$ level appears as a reflux-divergence around the projection of the $(\ell+1)$ grids, while the mismatch between the solution on the level ℓ and the coarser $\ell-1$ solution appears in the coarse-fine boundary condition. On the finest level, all we are doing is relaxing the correction to account for the mismatch due to the correction on the coarser levels. A pseudocode description of this algorithm is shown in Figure 3.14.

Convergence History

The convergence history for this algorithm is shown in Figure 3.15, which shows the L_1 norm of the composite residual vs. number of correction iterations for two-, three-, four-, and five-level solutions (a two-level solution has a base grid and one level of refinement). For this algorithm, the composite residual decreases monotonically with correction iterations. For each case, the residual drops off at a slower rate for $(\ell^{max}-2)$ iterations (which is (*number of coarse-fine interfaces*) -1), and then drops off very rapidly down to roundoff. This is apparently due to the need to correct for the effects of the correction on the composite solution. Figure 3.16 shows the residual for the three-level case.

3.3.3 Top-Down Iteration

In this version of the algorithm, the level solves are done as before, but the corrections are done starting at the finest level and proceeding down to the coarsest level. This means that

```
BottomUp($\epsilon$)
    do $\ell = 0, \ell_{max}$
        solve $L^\ell(\phi^\ell, \phi^{\ell-1}) = \rho^\ell$,     $\phi^\ell = I(\phi^\ell, \phi^{\ell-1})$  on  $\partial\Omega^{\ell/\ell-1}$
        if ($\ell < \ell_{max}$)
            $\delta\phi^{\ell+1} = -G^\ell\phi^\ell$  on  $\partial\Omega^{\ell/\ell+1}$
        if ($\ell \neq 0$)
            $\delta\phi^\ell = \delta\phi^\ell + \langle G^\ell\phi^\ell\rangle$  on  $\partial\Omega^{\ell/\ell-1}$
        $\delta e^\ell = \delta\phi^\ell$
    end do

    Res = $\rho - L^{comp}(\phi)$

    while ($\|\text{Res}\| < \epsilon\|\rho\|$)  do:
        do $\ell = 0, \ell_{max}$
            solve $L^\ell(e^\ell, e^{\ell-1}) = D_R(\delta e^{\ell+1})$,     $e = I(e^\ell, e^{\ell-1})$  on  $\partial\Omega^{\ell/\ell-1}$
            if ($\ell < \ell_{max}$)
                $\delta e^{\ell+1} = \delta\phi^{\ell+1} - G^\ell e^\ell$  on  $\partial\Omega^{\ell/\ell+1}$
            if ($\ell \neq 0$)
                $\delta e^\ell = \delta e^\ell + \langle G^\ell e^\ell\rangle$  on  $\partial\Omega^{\ell/\ell-1}$
        end do
        Res = $\rho - L^{comp}(\phi)$
    end while
end BottomUp
```

Figure 3.14: Bottom-up iteration algorithm

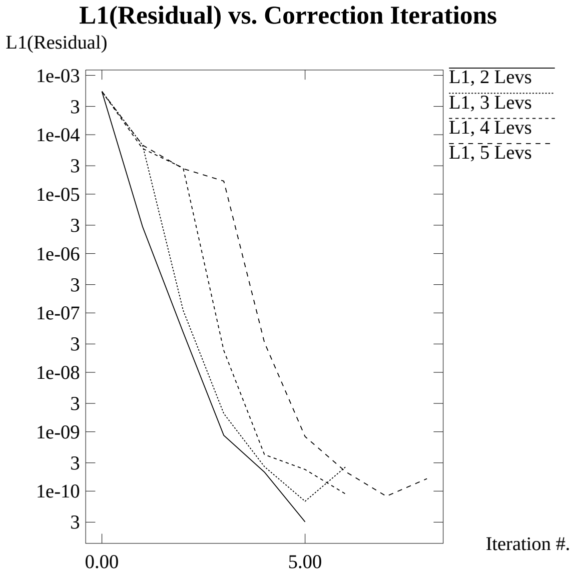
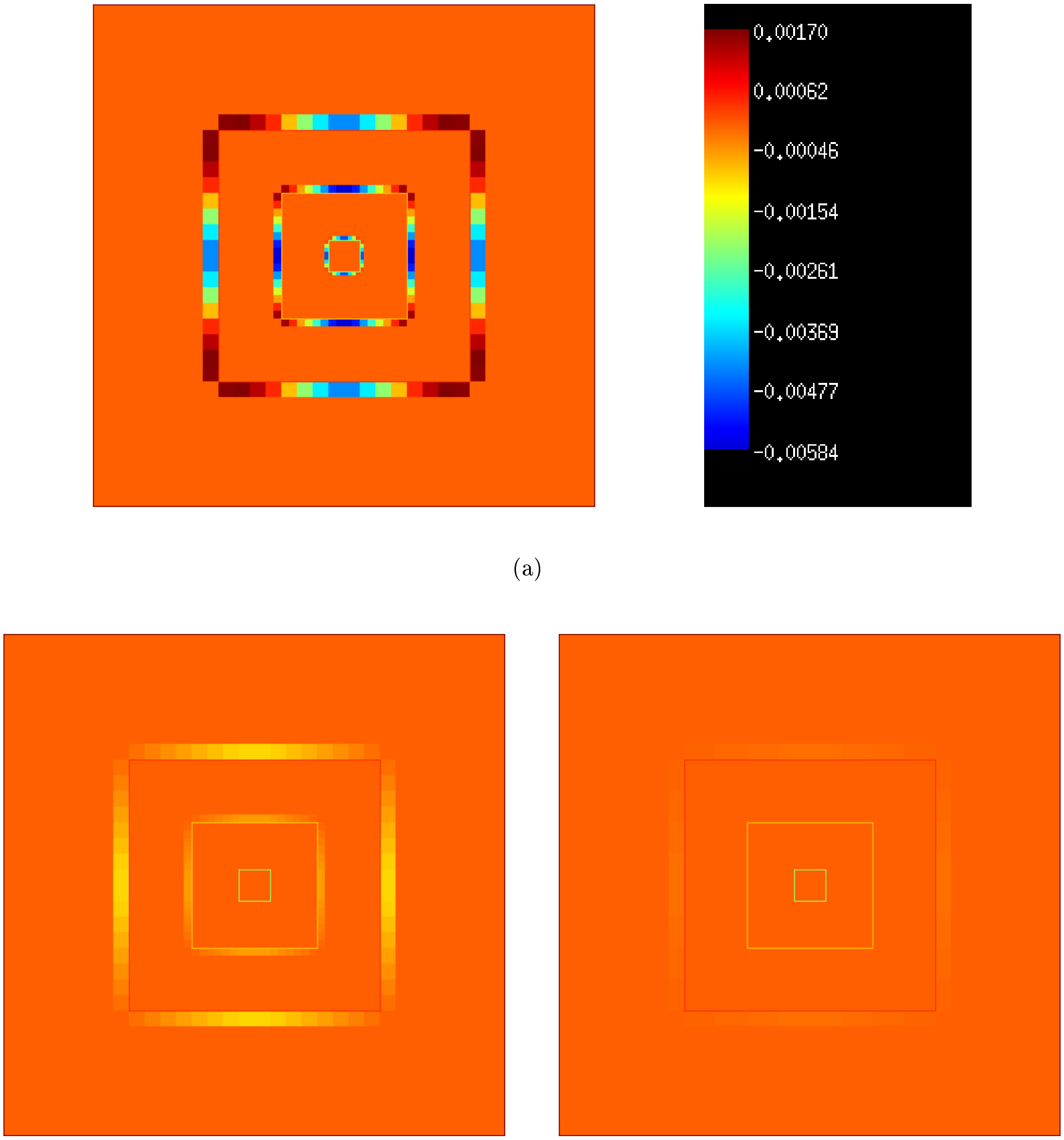


Figure 3.15: Convergence history – bottom-up iterations

the coarse-level corrections used for the coarse-fine boundary condition for the current correction is lagged behind the current correction. On the finest level, the initial correction does nothing, since the coarse correction is initially 0, and there is no residual induced from a finer level. Then, the initial correction on the coarser levels is solely due to the mismatch in ϕ (the level-solve solution) at the $\ell/\ell+1$ interface. Subsequent corrections on a level ℓ then account for the current mismatch with the finer level $\ell+1$ as well as the lagged mismatch with the coarser level $\ell-1$. A pseudocode description of this algorithm is shown in Figure 3.3.3.

Convergence History

The convergence history for this algorithm is shown in Figure 3.18, which shows the L_1 norm of the composite residual vs. number of corrections for two-, three-, four-, and five-level solutions. Note that, as opposed to the results shown in Figure 3.15, the residual initially *rises*,



(a)

(b)

(c)

Figure 3.16: Residual for bottom up iteration: (a) initial residual (after level solves), (b) after 1 multigrid correction iteration, and (c) after 2 multigrid correction iterations

```
TopDown($\epsilon$)
    do $\ell = 0,\ \ell_{max}$
        solve $L^\ell(\phi^\ell, \phi^{\ell-1}) = \rho^\ell$,    $\phi^\ell = I(\phi^\ell, \phi^{\ell-1})$ on $\partial\Omega^{\ell/\ell-1}$
        if $(\ell < \ell_{max})$
            $\delta\phi^{\ell+1} = -G\phi^\ell$ on $\partial\Omega^{\ell/\ell+1}$
        if $(\ell \neq 0)$
            $\delta\phi^\ell = \delta\phi^\ell + \langle G^\ell \phi^\ell \rangle$ on $\partial\Omega^{\ell/\ell-1}$
        $\delta e^\ell = \delta\phi^\ell$
    end do

    $\text{Res} = \rho - L^{comp}(\phi)$

    while $(\|\text{Res}\| < \epsilon\|\rho\|)$ do:
        do $\ell = \ell_{max}, 0$
            solve $L^\ell e = D_R(\delta e^{\ell+1})$,    $e = I(e^\ell, e^{\ell-1})$ on $\partial\Omega^{\ell/\ell-1}$
            if $(\ell \neq 0)$
                $\delta e^\ell = \delta\phi^\ell + \langle G^\ell e^\ell \rangle - G^\ell e^{\ell-1}$ on $\partial\Omega^{\ell/\ell-1}$
        end do
        $\text{Res} = \rho - L^{comp}(\phi)$
    end while
end TopDown
```

Figure 3.17: Top-down iteration algorithm

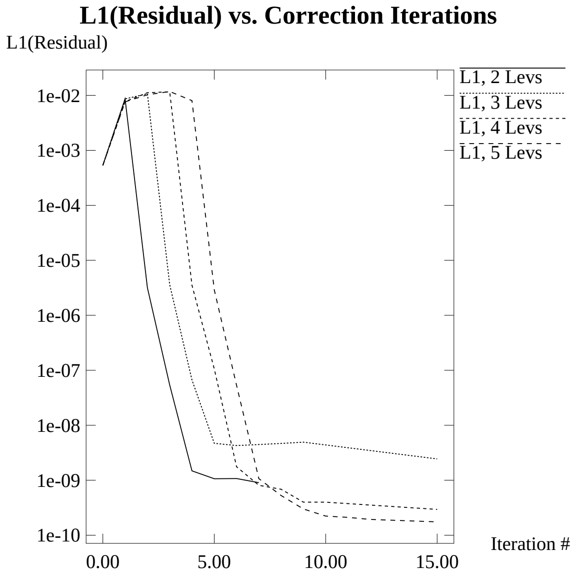


Figure 3.18: Convergence history – top-down iterations

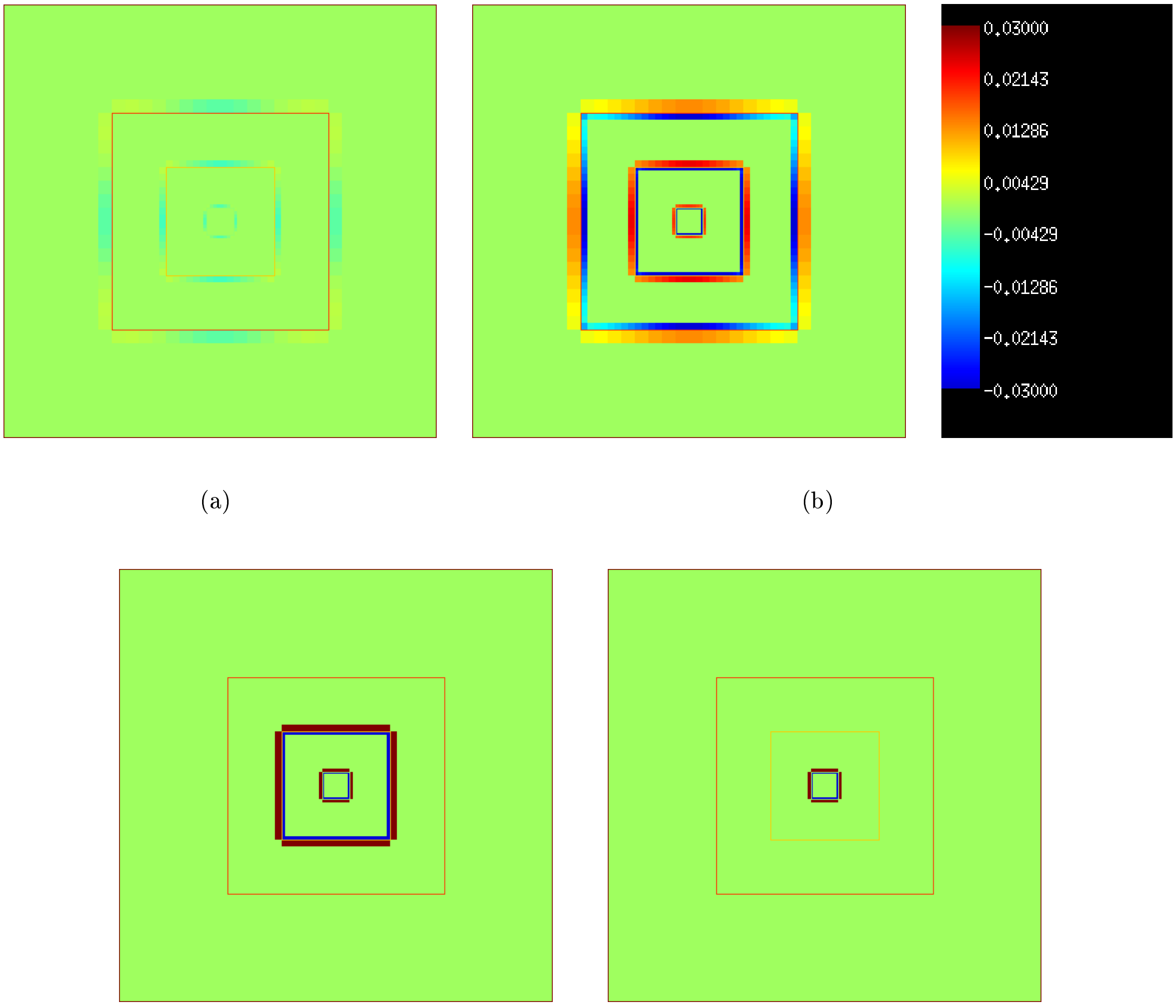
stays relatively constant, and then begins a rapid decrease to roundoff after ℓ_{max} iterations.

As can be seen in Figure 3.19, it appears that the residual first increases at all coarse-fine interfaces, and then the large residual is eliminated at the level 0/1 interface, then the level 1/2 interface, and so on up the hierarchy until all the large coarse-fine interface error has been eliminated.

This slowdown in convergence is most likely due do the lagged nature of the corrections. One way to think about this is that there is not really one correction, but a series of corrections $e_0, e_1, e_2, ...$ and the total solution is equal to the result from the level solves ϕ plus the sum of the corrections:

$$\Phi = \phi + \sum_{n=0}^{n=n_{max}} e_n \tag{3.46}$$

Reiterating (3.45), the correction should satisfy:



(a) (b)

(c) (d)

Figure 3.19: Residual for top down iteration: (a) initial residual after level solves, (b) after 1 multigrid correction iteration, (c) after 2 multigrid correction iterations, and (d) after 4 multigrid correction iterations.

$$\begin{aligned} L^\ell e &= -D_R(\delta\phi^{\ell+1}) - D_R(\delta e^{\ell+1}) \\ e &= I(e^\ell, e^{\ell-1}) \end{aligned}$$

With this algorithm, the correction e_n is actually satisfying:

$$\begin{aligned} L^\ell e_n &= -D_R(\delta\phi^{\ell+1}) - D_R(\delta_F e_n^{\ell+1}) - \sum_{k=0}^{k=n-1} [D_R(\delta e_k^{\ell+1}) + L^\ell e_k] \\ e_n &= I(e_n^\ell, e_{n-1}^{\ell-1}) \end{aligned} \tag{3.47}$$

The notation $\delta_F e_n^{\ell+1}$ refers to the normal flux register, but with the fine level component *only*, due to the lagged nature of the correction:

$$\delta_F e_n^{\ell+1} =< G^\ell e_n^{\ell+1} > \quad \text{on} \quad \partial\Omega^{\ell/\ell+1}. \tag{3.48}$$

Note also the mismatch in the coarse-fine boundary condition.

The first correction, e_0, solves only for the mismatch in ϕ between the ℓ and $\ell+1$ levels, along with the effect of the fine level correction on the solution. In subsequent iterations, the equation being solved is:

$$L^\ell e_n = -D_R(\delta_C e_{n-1}^{\ell+1}) - D_R(\delta_F e_n^{\ell+1}) e_n = I(e_n^\ell, e_{n-1}^{\ell-1}) \quad \text{on} \quad \partial\Omega^{\ell/\ell-1}, \tag{3.49}$$

where $\delta_C e_{n-1}^{\ell+1}$ is the coarse level contribution to the level $\ell/\ell+1$ flux register:

$$\delta_C e_n^{\ell+1} = G^\ell e_n^\ell \quad \text{on} \quad \partial\Omega^{\ell/\ell+1}. \tag{3.50}$$

So, the effect of the large coarse-fine error is propagated down to the level 0/1 interface, where it is eliminated, and then the level 1/2 interface can be updated, and so on.

3.4 Convergence and Errors

We also looked at the global convergence of the multilevel Poisson solvers described in this chapter. Because the composite operators, residual, and convergence criteria were defined in the same way for each method, we expect that the accuracy and errors in each will be comparable. So, while the analysis in this section was conducted using the bottom-up level solve/correction algorithm of Section 3.3.2 (because that is what we were working on when this work was done), it should be applicable to the standard multilevel solution algorithm as well.

3.4.1 Convergence

To obtain convergence results, we solved Poisson's equation for a problem for which we have an exact solution. This enabled us to better look at errors in the solution and their behavior as the grids were refined. The sample problem used was a single quartic source, with Dirichlet physical boundary conditions set to be the exact solution on $\partial\Omega$ if the problem was being solved in an infinite domain using the higher-order ghost-cell discretization (2.3). For these convergence studies, the strategy was to fix a certain number of levels (in this case we looked at two-level solutions – base level + one level of refinement), and then let the Richardson extrapolation error estimator (Section 5.3) generate grids adaptively based on its truncation error estimates. As we refine the base grids, we also scale the regridding error tolerance for consistency. Since we expect the algorithm to be $O(h^2)$, we scale the tolerance in the same way; for example, the tolerance for a 128×128 base grid would be $\frac{1}{4^2}$ the tolerance of the 32×32 case.

Quantities that we looked at were:

- Error:

$$\epsilon = \phi_{exact} - \phi$$

- Truncation Error:

$$\begin{aligned} \tau &= L(\phi_{exact}) - L(\phi) \\ &= L(\phi_{exact}) - \rho \end{aligned} \tag{3.51}$$

- Boundary truncation error, τ_{bnd}, the truncation error on cells adjacent to coarse-fine interfaces (on both the fine and coarse sides of the interface).

- Internal truncation error, τ_{int}: the complement to τ_{bnd}

$$\tau_{int} = \tau - \tau_{bnd} \tag{3.52}$$

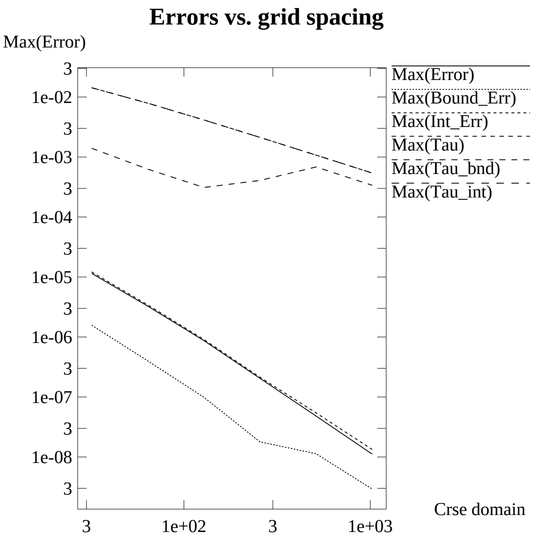
- Boundary Error: The error in the solution which is a result of truncation error at the coarse-fine interfaces. To compute this, we solve the equation:

$$\begin{aligned} L(\epsilon_{bnd}) &= \tau_{bnd}, \\ \epsilon_{bnd} &= 0 \text{ on } \partial\Omega \end{aligned} \tag{3.53}$$

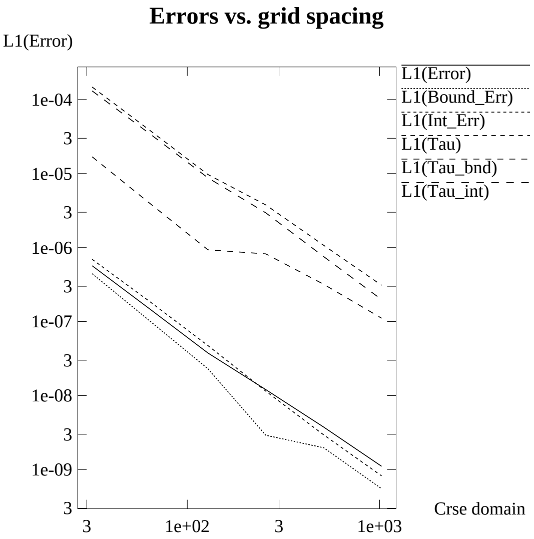
- Internal error: The complement to ϵ_{bnd} (this also includes the error due to physical boundary conditions).

$$\epsilon_{int} = \epsilon - \epsilon_{bnd} \tag{3.54}$$

Plots of these errors vs. coarse grid spacing are shown in Figure 3.20 in L_∞ and L_1 norms. The "bump" in the error after the 128×128 base grid occurs due to a change in the grid configuration.



(a)



(b)

Figure 3.20: Errors vs. grid resolution. Errors in (a) L_∞Norm, and (b) L_1 Norm

Up to this point, the fine grids occupy a corner in the domain, with both the top and right sides of the refined patch abutting the physical boundary. As the base grids get finer, the fine patch separates from the physical boundary, so that it has coarse-fine interfaces on all four sides. It is worth noting that this rearrangement of the grids has little, if any, effect on the global error and truncation errors – it just leads to a redistribution of the error between "internal" and "boundary" components.

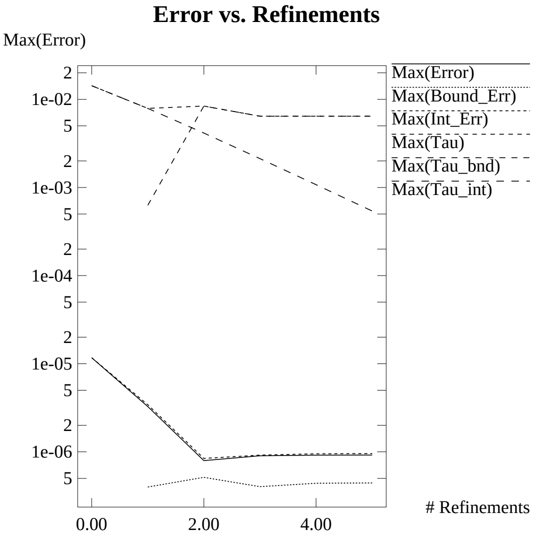
As expected, the L_∞ norm of the global error is $O(h^2)$, as is ϵ_{int}. Also as expected, the L_∞ norm of the truncation error is $O(h)$, due in this case to the truncation error induced at the physical boundaries. The L_1 convergence of this algorithm appears to be between $O(h^{1.6})$ and $O(h^2)$.

3.4.2 Effects of Local Refinement

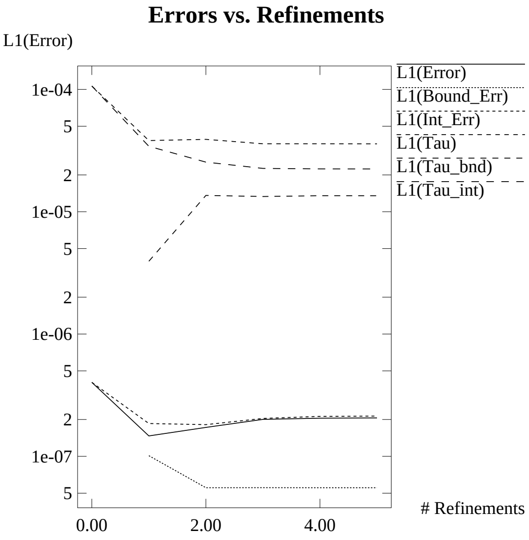
Another interesting result was obtained by taking the 64×64 base grid case and allowing the grid generator to generate as many grids as it deemed necessary. The errors as a function of number of refinements is shown in Figure 3.21 for the L_∞ and L_1 norms. It is apparent that refinement is only beneficial for one or two levels of refinement, after which the error no longer decreases, and actually increases slightly. We believe this to be the result of two tendencies, both due to the increasingly singular nature of the solution as we refine.

First, we are refining smaller and smaller portions of the domain, so it is apparent that local improvement of the solution in a tiny portion of the domain will have little to no real effect on the global solution.

Second, although we are improving the accuracy of the solution on the interior of the fine grids (note the steady improvement of the internal component of the truncation error), we are balancing this with the creation of new coarse-fine interfaces, with their associated $O(h)$ error.



(a)



(b)

Figure 3.21: Effects of local refinement. Errors in (a) L_∞ Norm, and (b) L_1 Norm

The effects of this increased boundary error are most apparent by looking at the truncation error – although the internal component decreases steadily as more fine levels are added, the associated increased boundary error causes the total truncation error to remain constant.

This has implications on our regridding and error-estimation strategies. For instance, Trompert and Verwer [67] include the increased error due to interpolation errors on the coarse-fine boundary in their regridding criterion, and point out that it is best to place coarse-fine interfaces where the solution is not strongly varying, to minimize the effects of coarse-fine interface errors. Also, Propp [51] presents a flux-based Richardson extrapolation error estimation method which accounts for the surface to volume ratio of the refined grids.

Chapter 4

Adaptive Projection Algorithm

This chapter will describe the extension of the single-grid projection algorithm defined in Chapter 2 to AMR. We will extend the adaptive algorithm developed by Berger and Colella [18] for hyperbolic conservation laws to construct an AMR projection method for solving the incompressible Euler equations. For context, a brief review of the algorithm in [18] is in order.

4.1 AMR for Hyperbolic Conservation Laws

Berger and Colella [18] developed a locally adaptive methodology for solving hyperbolic conservation laws. Their method refined in time as well as space, and maintained conservation at coarse-fine interfaces.

4.1.1 Conservation Laws

In [18], the equation being solved is a system of hyperbolic conservation laws, which have the form:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} f_x + \frac{\partial}{\partial y} f_y = 0, \tag{4.1}$$

where u is the conserved quantity, and $\mathbf{f} = (f_x, f_y)^T$ is the flux function. Integrating (4.1), using the divergence theorem:

$$\begin{aligned} \frac{\partial}{\partial t}\int_\Omega u dV &= -\int_\Omega \nabla \cdot \mathbf{f}(u) dV \\ &= -\int_{\partial\Omega} \mathbf{f}(u) \cdot \mathbf{n} dS, \end{aligned} \tag{4.2}$$

where $\mathbf{n}$ is the normal of the boundary of Ω. So, the change in the integral of u over any given domain will be equal to the integrated fluxes through the boundary of the area.

Using a conservative method will guarantee that u satisfies a discrete analog of (4.2). In two dimensions, a conservative method will take the form:

$$\begin{aligned} U^{n+1} &= U^n - \frac{\Delta t}{\Delta x}\left(F^{n+\frac{1}{2}}_{x,i+\frac{1}{2},j} - F^{n+\frac{1}{2}}_{x,i-\frac{1}{2},j}\right) - \frac{\Delta t}{\Delta y}\left(F^{n+\frac{1}{2}}_{y,i,j+\frac{1}{2}} - F^{n+\frac{1}{2}}_{y,i,j-\frac{1}{2}}\right), \\ &= U^n - \Delta t D(F^{n+\frac{1}{2}}) \end{aligned} \tag{4.3}$$

where $U^n_{i,j}$ is a cell-centered approximation to the cell average of u, $\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} u(x,y,t^n) dy dx$, and F_x and F_y are numerical approximations to f_x and f_y, averaged over the cell-edges and over the timestep. Because we use the same edge fluxes to update the cells on both sides of each edge, U is conserved. In a numerical scheme, conservation implies that for any cell, or group of cells, the integrated change in U over a time Δt will be the sum of the numerical fluxes $\mathbf{F}$ through the cell-edges around the cells:

$$\sum_\Omega U^{n+1}_{i,j} = \sum_\Omega U^n_{i,j} - \Delta t \sum_{\partial\Omega} F \cdot \mathbf{n}, \tag{4.4}$$

where $\sum_\Omega$ represents the sum over all the cells (i,j) in a region Ω, and $\sum_{\partial\Omega}$ represents a sum over all of the cell edges which also make up the boundary $\partial\Omega$. (For more background on conservative methods for hyperbolic conservation laws, see LeVeque [46].)

To advance this equation on a single grid, we would follow the procedure outlined in

```
AdvanceSoln($t, \Delta t$)
    FillGhostCells($t, U(t)$)
    Compute $\mathbf{F} = (F_x^{n+\frac{1}{2}}, F_y^{n+\frac{1}{2}})^T$
    $U(t+\Delta t) = U(t) - \frac{\Delta t}{\Delta x}\left(F^{n+\frac{1}{2}}_{x,i+\frac{1}{2},j} - F^{n+\frac{1}{2}}_{x,i-\frac{1}{2},j}\right) - \frac{\Delta t}{\Delta y}\left(F^{n+\frac{1}{2}}_{y,i,j+\frac{1}{2}} - F^{n+\frac{1}{2}}_{y,i,j-\frac{1}{2}}\right)$
end AdvanceSoln
```

Figure 4.1: Single-grid update for hyperbolic conservation laws

Figure 4.1. First, we fill ghost cells around the physical domain with values which represent the appropriate physical boundary condition on U. Then, we step through all the edges in the domain, first computing the edge-centered fluxes $\mathbf{F^{n+\frac{1}{2}}} = (F_x^{n+\frac{1}{2}}, F_y^{n+\frac{1}{2}})^T$. Finally, we update U using the conservative update (4.3).

4.1.2 Adaptive Methodology

In [18], block-structured local refinement is employed – the adaptive hierarchy of refined grids used in that work is similar in structure to that described in Section 3.1. In this scheme, refinement is temporal as well as spatial – fine cells are advanced using a finer timestep than is used to advance coarser cells. The authors employ a recursive timestepping algorithm in which coarse levels are updated, followed by successively finer levels. Proper nesting of refined grids (see Section 3.1.1) ensures that interpolation of coarse-grid data can provide boundary conditions for the fine-grid updates.

Refinement in Time

Many implementations of time-dependent AMR algorithms, including those in [48, 67] advance all levels at the same global timestep. While this results in a simpler time-stepping algorithm, it is less efficient and less accurate due to the fact that the global timestep is restricted by the stability requirements of the finest cells.

For stability, most explicit time-dependent schemes must satisfy some form of a Courant-Friedrichs-Lewy (CFL) [31] condition,

$$\sigma = max(\frac{u}{\Delta x}, \frac{v}{\Delta y})\Delta t < C \tag{4.5}$$

where C is determined by the particular scheme being used. σ is known as the CFL number. For most explicit advection schemes, $C = 1$. Note that this requires that as the mesh spacing is decreased, there must be a corresponding decrease in the timestep.

When local refinement is used, different regions of the solution have different levels of spatial refinement. If all levels are advanced at the same timestep, the coarse levels will need to be advanced at a much finer timestep than would be dictated by the stability requirements of the coarse levels alone, in order to ensure stability at the finest levels. This results in more computational work being done on the coarse levels (where less resolution is required) than is necessary, and so is less efficient than we would like. Also, the advection schemes we are using are more accurate at moderate CFL numbers, and become more dispersive as the CFL number goes to zero [68].

For these reasons, when solving time-dependent equations with local refinement, we would like to refine in time as well as space. This is known as *subcycling*. Advancing the finer grids at a finer timestep ensures that the global timestep is not held hostage to the restrictive stability requirements of the finer grid.

In [18], finer levels are advanced at a finer timestep than coarser ones. If level $\ell + 1$ is a factor of n^{ℓ}_{ref} finer spatially than the coarser level ℓ, then the finer level will be advanced using a

```
CompositeTimeStep($t^{crse}, \Delta t^{crse}$)
    Advance $U^{crse}(t^{crse}) \rightarrow U^{crse}(t^{crse} + \Delta t^{crse})$
    for $n = 0, n_{ref} - 1$
        $\Delta t^{fine} = \frac{1}{n_{ref}} \Delta t^{crse}$
        $t^{fine} = t^{crse} + n\Delta t^{fine}$
        Advance $U^{fine}(t^{fine}) \rightarrow U^{fine}(t^{fine} + \Delta t^{fine})$
    end for
    synchronize($U^{crse}(t^{crse} + \Delta t^{crse}), U^{fine}(t^{crse} + \Delta t^{crse})$)
end CompositeTimeStep
```

Figure 4.2: Pseudocode for composite solution advance for two-level case

timestep which is a factor of n^{ℓ}_{ref} finer than the timestep on the coarser level:

$$\Delta t^{\ell+1} = \frac{1}{n^{\ell}_{ref}} \Delta t^{\ell}. \tag{4.6}$$

This results in a more efficient time-stepping procedure, since all levels are advanced using approximately the same CFL number.

Time-stepping Strategy

First, consider the two-level case, with one coarse and one fine level. Assume that we have a composite solution which is defined at time t^{crse}:

$$U^{comp}(t^{crse}) = \begin{cases} U^{crse}(t^{crse}) & \text{on } \Omega^{crse} \\ U^{fine}(t^{crse}) & \text{on } \Omega^{fine} \end{cases} \tag{4.7}$$

The goal of computation will be to advance the composite solution to a new time $t^{crse} + \Delta t^{crse}$.

The timestepping strategy employed in [18] is to first advance the coarse level from t^{crse} to $t^{crse} + \Delta t^{crse}$. (Figure 4.2) This coarse-grid update will be structured exactly the same as the single-grid update described in Figure 4.1. Specifically, we compute $\mathbf{F}^{\mathbf{n}+\frac{1}{2},\ell}$ on $\Omega^{\ell,*}$, and apply the level-divergence operator componentwise to obtain the update for U:

$$U := U - \Delta t D^{\ell} \mathbf{F}^{\ell}.$$

Then, the fine level will be updated n_{ref} times, with a timestep of $\Delta t^{fine} = \frac{1}{n_{ref}} \Delta t^{crse}$ in a similar way. Each fine-level update will be structured in the same way as Figure 4.1 for each fine-level grid (recall that the fine level Ω^{fine} is made up of a union of rectangular fine-level grids), and will advance the fine-level solution from t^{fine} to $t^{fine} + \Delta t^{fine}$. First, we will fill ghost cells around each fine-level grid with appropriate values. Then we can advance each fine-level grid independently, as if it were a single-grid solution. The only difference between the fine-grid updates and the single-grid update is that ghost cells may now represent boundary conditions from the coarse level or from another fine-level grid as well as physical boundary conditions. By using ghost cells to enforce appropriate boundary conditions for each fine-level grid, we can separate the details of the AMR implementation from the level update, and use essentially the same update for each grid as we used for the single-grid update. This enormously simplifies addition of AMR capabilities to existing algorithms.

In general, for the higher-order hyperbolic schemes we will use, we will need to fill a border of ghost cells more than one cell wide around each grid. When boundary conditions are computed, we fill enough ghost cells to complete the stencils for each cell in the valid domain on each grid.

Recall that the boundary of a fine-level grid can be either a physical boundary, a coarse-fine interface with the coarse level, or a fine-fine interface with another grid in the fine level (or some mixture of the three). Filling ghost cells where $\partial\Omega^{fine}$ is a physical boundary is straightforward, using the the standard ghost-cell formulation used in a single-grid update. Where $\partial\Omega^{fine}$ is a coarse-fine interface with the coarse level, coarse-grid solution values U^{crse} are linearly interpolated in time to $U^{crse}(t^{fine})$, which is possible because the coarse level has already been updated to $t^{crse} + \Delta t^{crse}$.

Then $U^{crse}(t^{fine})$ is interpolated in space using conservative linear interpolation to fill the ghost cells around the fine level. Finally, where a fine-level grid abuts another grid at the same level of refinement, the ghost cells are filled by simply copying $U^{fine}(t^{fine})$ from the interiors of the other fine-level grids. By copying values from the interiors of each grid at the current level, we can make the interfaces between grids seamless, which will make the solution independent of how the refined domain was decomposed into constituent rectangular grids, which is an important property of the algorithm.

The fine grid solution will be computed for $U(t^{fine}), t^{fine} \in \{t^{crse} + k\Delta t^{fine}\}_{k=0}^{n_{ref}}$. After n_{ref} timesteps, the fine level will reach the same time as the coarse level, $t^{crse} + \Delta t^{crse}$. At this point, the coarse and fine solutions must be brought into agreement, a process we will call *synchronization*. In [18], synchronization has two goals. First, we would like to use the more accurate fine-level solution wherever possible, and we also need to ensure that the advance of the composite solution from t^{crse} to t^{fine} is conservative.

Synchronization of the coarse and fine solutions for the hyperbolic problem consists of two steps. First, the coarse-grid solution at $t^{crse} + \Delta t^{crse}$ is replaced where possible by the averaged fine-grid solution:

$$U^{crse}(t^{sync})_{i,j} = Av(U^{fine}(t^{sync}))_{i,j} \quad \text{on } \mathrm{P}(\Omega^{\mathrm{fine}}) \tag{4.8}$$

where t^{sync} is the new time which both the coarse and fine solutions have reached.

Then, the coarse-grid solution is corrected to ensure conservation. For conservation, the flux out of the fine cells across the coarse-fine interface must be the same as the flux into the coarse cells through the interface. For example, consider a case where the coarse-fine interface is to the left of cell (i, j) (Figure 4.3). In this case, we used $F^{crse}_{x,i-\frac{1}{2},j}$ to compute the update for cell (i, j).

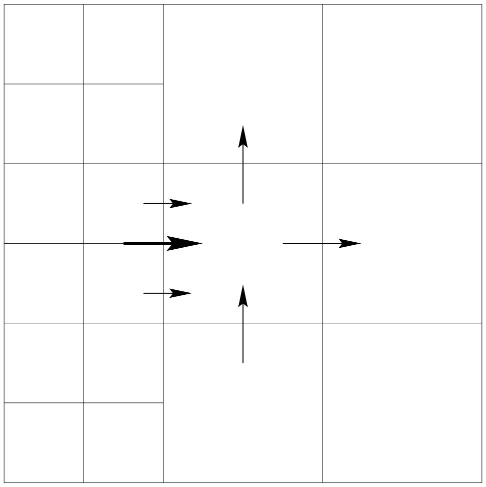


Figure 4.3: Coarse and fine fluxes across the coarse-fine interface.

However, on the fine side of the coarse-fine interface, the flux across the $(i-\frac{1}{2},j)$ edge during the same interval was $\frac{1}{n_{ref}}\sum\langle F^{fine}\rangle$, where the $\langle - \rangle$ notation represents a spatial average over the fine edges which overlay the coarse-cell edge $(i-\frac{1}{2},j)$. The sum is over the subcycled fine-level timesteps, and the factor $\frac{1}{n_{ref}}$ accounts for the fact that $\Delta t^{fine} = \frac{1}{n_{ref}}\Delta t^{crse}$. For conservation, we require that the same fluxes be used in the updates on both the coarse and fine side of the interface. As a rule, we consider the fine-grid information to be more accurate, so we would like to update the coarse-grid cells adjacent to the interface using the fluxes computed during the fine-grid updates. For cell (i,j), this means that the update (4.3) must be modified to use the fine-grid computed fluxes:

$$U^{n+1,crse}_{i,j} = U^{n,crse}_{i,j} - \frac{\Delta t^{crse}}{\Delta x}\left(F^{crse}_{x,i+\frac{1}{2},j} - \frac{1}{n_{ref}}\sum\langle F^{fine}_{x}\rangle_{i-\frac{1}{2},j}\right) - \frac{\Delta t^{crse}}{\Delta y}\left(F^{crse}_{y,i,j+\frac{1}{2}} - F^{crse}_{y,i,j-\frac{1}{2}}\right). \quad (4.9)$$

Adding and subtracting $\frac{dt^{crse}}{\Delta x^{crse}}F^{crse}_{x,i-\frac{1}{2},j}$ from (4.9), we get

$$U^{n+1}_{i,j} = U^{n,crse}_{i,j} - \frac{\Delta t^{crse}}{\Delta x}\left(F^{crse}_{x,i+\frac{1}{2},j} - F^{crse}_{x,i,j-\frac{1}{2}}\right) - \frac{\Delta t^{crse}}{\Delta y}\left(F^{crse}_{y,i,j+\frac{1}{2}} - F^{crse}_{y,i,j-\frac{1}{2}}\right) \quad (4.10)$$

$$-\frac{\Delta t^{crse}}{\Delta x}\Big(\frac{1}{n_{ref}}\sum\langle F_x^{fine}\rangle_{i-\frac{1}{2},j} - F^{crse}_{x,i,j-\frac{1}{2}}\Big)$$

Notice that we have recovered the original single-grid update (4.3) with a correction for the effect of the fine grid. This fine-grid correction can be expressed using the flux register and reflux-divergence notation of Section 3.1.2. If we define the flux register δF^{fine} as the difference in the coarse and fine fluxes:

$$\delta F^{fine} = \frac{1}{n_{ref}}\sum\langle F^{fine}\rangle - F^{crse} \tag{4.11}$$

and use the reflux divergence operator D_R defined in Section 3.1.2, then (4.10) can be written:

$$\begin{aligned} U^{n+1}_{i,j} &= U^{n,crse}_{i,j} - \frac{\Delta t^{crse}}{\Delta x}\Big(F^{crse}_{x,i+\frac{1}{2},j} - F^{crse}_{x,i,j-\frac{1}{2}}\Big) - \frac{\Delta t^{crse}}{\Delta y}\Big(F^{crse}_{y,i,j+\frac{1}{2}} - F^{crse}_{y,i,j-\frac{1}{2}}\Big) \qquad (4.12)\\ &\quad -\Delta t^{crse} D_R(\delta F^{fine}). \\ &= U^{n,crse}_{i,j} - \Delta t^{crse} D^{crse}(F^{crse})_{i,j} - \Delta t^{crse} D_R(\delta F^{fine})_{i,j} \end{aligned}$$

We will call the operation of correcting the coarse grid solution by subtracting the reflux-divergence of the mismatch in fluxes *refluxing.*

So, the coarse grid solution can be corrected to enforce the flux-matching condition required by conservation by a simple refluxing operation, which can be performed separately from the coarse-grid update. In the case of refinement in time, as in the algorithm of [18], the refluxing operation is performed during the synchronization step, after al step, after all of the relevant coarse and fine updates have been performed.

Once the fine solution has been averaged onto the coarse level and the coarse fluxes have been corrected by refluxing, then the advance of the composite solution from time t^{crse} to $t^{crse} + \Delta t^{crse}$ is complete.

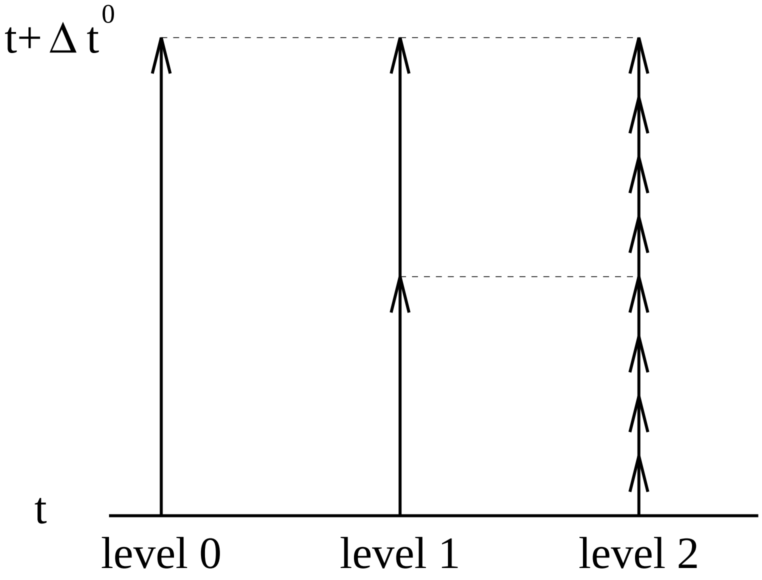


Figure 4.4: Schematic of subcycled timestep

4.1.3 Recursive Timestepping Algorithm

The two-level algorithm in the previous section is generalized in [18] for any number of levels by redefining the algorithm as a series of recursive single-level advances. Figure 4.4 shows a sample global timestep with 2 levels of refinement; the first level of refinement is a factor of 2 finer than the base level, while level 2 is a factor of 4 finer than level 1. To update the composite solution, we do a level advance for level 0 from time t^0 to time $t^0 + \Delta t^0$. Then, we perform a level advance on level 1, with $\Delta t^1 = \frac{1}{2}\Delta t^0$. Since the timestepping is recursive, we then will do 4 level advances on level 2, each with a timestep of $\Delta t^2 = \frac{1}{4}\Delta t^1$. This will bring level 2 and level 1 to the same time. We then synchronize levels 1 and 2. Once level 1 and level 2 have been synchronized, level 1 can be advanced again. Once again, $\Delta t^1 = \frac{1}{2}\Delta t^0$. Then, level 2 is advanced four times with $\Delta t^2 = \frac{1}{4}\Delta t^1$. At this point, all the levels have reached the new coarse time. So, we then synchronize all of the

```
LevelAdvance($\ell, t^\ell, \Delta t^\ell$)
    fillGhostCells($t$)
    compute $F_x^\ell, F_y^\ell$
    $U^\ell(t^\ell + \Delta t^\ell)_{i,j} = U^\ell(t^\ell)_{i,j} - \frac{\Delta t^\ell}{\Delta x}\left(F^\ell_{x,i+\frac{1}{2},j} - F^\ell_{y,i-\frac{1}{2},j}\right) - \frac{\Delta t^\ell}{\Delta y}\left(F^\ell_{y,i,j+\frac{1}{2}} - F^\ell_{\ell,i,j-\frac{1}{2}}\right)$
    Update Flux Registers:
        if ($\ell < \ell_{max}$) $\delta F^{\ell+1} = -\mathbf{F}^\ell \cdot \mathbf{n}_{CF}^{\ell+1}$    on $\partial\Omega^{\ell+1}$
        if ($\ell > 0$) $\delta F^\ell = \delta F^\ell + \frac{1}{n_{ref}^{\ell-1}}\langle \mathbf{F}^\ell \cdot \mathbf{n}_{CF}^\ell \rangle$    on $\partial\Omega^\ell$
    if ($\ell < \ell_{max}$) then
        for $n = 0, n_{ref}^\ell - 1$
            $\Delta t^{\ell+1} = \frac{1}{n_{ref}^\ell}\Delta t^\ell$
            $t^{\ell+1} = t^\ell + n\Delta t^{\ell+1}$
            LevelAdvance($\ell + 1, t^{\ell+1}, \Delta t^{\ell+1}$)
        end for
    $U^\ell(t^\ell + \Delta t^\ell) := U^\ell(t^\ell + \Delta t^\ell) - \Delta t D_R(\delta F^{\ell+1})$

    end if
end LevelAdvance
```

Figure 4.5: Pseudocode for recursive timestep used for hyperbolic conservation laws in Berger and Colella

levels, which will result in the final composite solution at time $t^0 + \Delta t^0$.

The function *LevelAdvance*$(\ell, t^\ell, \Delta t^\ell)$ (Figure 4.5) will advance the level ℓ solution from time t^ℓ to time $t^\ell + \Delta t^\ell$. Because this function is recursive, all finer levels (which initially will also be at time t^ℓ) will also be advanced to the new time, and the appropriate synchronization operations will be performed so that the entire composite solution for levels $\ell..\ell_{max}$ will be advanced to the new time. So, to advance the entire solution from time t^0 to time $t^0 + \Delta t^0$, we call *LevelAdvance*$(0, t^0, \Delta t^0)$, which will advance the entire composite solution through a series of recursive level advances.

Once the ghost cells have been filled in the same way as the two-level case, each grid in level

ℓ can be updated independently, using the same update method as in the single grid case. First, we compute the fluxes F_x and F_y for each edge. Then, we compute the update to U^ℓ using the single-grid update defined by (4.3). At this point, the appropriate flux registers are updated. If a finer level exists, then the level $\ell+1$ flux register is initialized with the coarse level flux $(-\mathbf{F}^\ell \cdot \mathbf{n}_{CF}^{\ell+1})$, where $\mathbf{n}_{CF}^{\ell+1}$ is the local normal of the coarse-fine interface between levels ℓ and $\ell+1$. If a coarser level exists, then the level ℓ flux register is incremented with the fine-level flux from this timestep.

Once all the cells in level ℓ have been updated to time $t^\ell + \Delta t^\ell$, we can recursively advance any finer levels, using the same timestepping procedure. The finer level $\ell+1$ is advanced n_{ref}^ℓ times, starting at the level ℓ initial time t^ℓ. The fine timestep will be $\Delta t^{\ell+1} = \frac{1}{n_{ref}^\ell}\Delta t^\ell$.

Once the fine level has been advanced n_{ref}^ℓ times, it has reached the same time as the level ℓ solution, which is $t^\ell + \Delta t^\ell$. At this time, the solutions on levels ℓ and $\ell+1$ are brought into agreement: the level ℓ solution $U^\ell(t^\ell + \Delta t^\ell)$ is replaced by the averaged fine solution $U^{\ell+1}(t^\ell + \Delta t^\ell)$ wherever level ℓ is covered by refinement, and the level ℓ solution is corrected by refluxing the mismatch of fine and coarse fluxes to ensure conservation.

4.2 Multilevel Discretization of the Incompressible Euler equations

We would like to extend the AMR methodology developed in the previous section to the solution of the incompressible Euler equations by extending the single-grid algorithm presented in Section 2.6. In the single-grid algorithm, we advance the velocity field $\mathbf{u}$ and a passively advected scalar field s from time t^n to time $t^n + \Delta t$. Also in that algorithm, a lagged pressure field $p^{n+\frac{1}{2}}$ is computed to enforce the incompressibility constraint. As in the previous section, we will define a solution on each level $\mathbf{u}^\ell, s^\ell$ and the associated pressure on each level π^ℓ. Also, as in the previous section, we will subcycle in time, so during a level ℓ timestep from t^ℓ to $t^\ell + \Delta t^\ell$, the level $\ell+1$ solution will be advanced n^ℓ_{ref} times. So, in a single level 0 timestep, we will compute solution values for each level $\ell > 0$ at the following times:

$$\mathbf{u}^\ell(t^\ell), s^\ell(t^\ell), t^\ell \in \{t^{\ell-1} + k\Delta t^\ell\}_{k=0}^{n^{\ell-1}_{ref}-1}. \tag{4.13}$$

Because the pressure is lagged in this algorithm, it will also be defined at lagged times on each level, or

$$\pi^\ell(t^\ell), t^\ell \in \{t^{\ell-1} + (k - \frac{1}{2})\Delta t^\ell\}_{k=1}^{n^{\ell-1}_{ref}}. \tag{4.14}$$

4.2.1 Level Algorithm

To extend the recursive subcycled algorithm of Section 4.1.3 to the projection algorithm described in Section 2.6, we will first need to express the single-grid projection algorithm as a level update which will advance the level ℓ solution from time t^ℓ to $t^\ell + \Delta t^\ell$. This is straightforward. A brief outline of this level update is as follows:

1. Compute advection velocities $\mathbf{u}^{AD,\ell}$ as in Section 2.6.1, including level ℓ edge-centered (MAC) projection for advection velocities.

2. Compute advective update for scalar:
   $s^\ell(t^\ell + \Delta t^\ell)_{i,j} := s^\ell(t^\ell) - \frac{\Delta t^\ell}{\Delta x^\ell}\left(F^{S,\ell}_{i+\frac{1}{2},j} - F^{S,\ell}_{i-\frac{1}{2},j}\right) - \frac{\Delta t^\ell}{\Delta y^\ell}\left(F^{S,\ell}_{i,j+\frac{1}{2}} - F^{S,\ell}_{i,j-\frac{1}{2}}\right)$

3. Compute intermediate velocity field $\mathbf{u}^{**,\ell}$:
   $\mathbf{u}^{**,\ell}_{i,j} := \mathbf{u}^\ell(t^\ell)_{i,j} - \Delta t^\ell[(\mathbf{u}\cdot\nabla)\mathbf{u}]^{n+\frac{1}{2},\ell}_{i,j}$

4. Project intermediate velocity field to enforce divergence constraint:

   Solve $L^\ell \pi^\ell (t^\ell + \frac{\Delta t^\ell}{2}) = D^{CC,\ell} \mathbf{u}^{**,\ell}$

   $\mathbf{u}^\ell (t^\ell + \Delta t^\ell) := \mathbf{u}^{**,\ell} - \Delta t^\ell G^{CC,\ell} \pi^\ell (t^\ell + \frac{\Delta t^\ell}{2})$

4.2.2 Level Operators

The outline in the previous section left open the issue of extending the cell-centered operators G^{CC} and D^{CC} defined in Section 2.5.2 to a level-operator formulation. In most cases, the level operator will simply be the corresponding single-grid operator, with a suitable coarse-fine boundary condition for use when the normal stencils cross a coarse-fine interface with level $\ell-1$. When defining coarse-fine boundary conditions for these operators, redefining them as edge-centered operators with appropriate cell-to-edge and edge-to-cell averaging (equations (2.62) and (2.63)) will prove useful. As in Section 4.1, use of ghost cells around each fine grid will simplify the application of boundary conditions by separating the boundary conditions from the operator discretization.

Gradient

We first define the level operator version of the edge-centered gradient, G^ℓ, which we will then extend to the cell-centered operator $G^{CC,\ell}$ through the use of (2.63), repeated here for convenience:

$$G^{CC} \phi = Av^{E \rightarrow C} G \phi. \tag{4.15}$$

G^ℓ will be the level-operator version of the edge-centered gradient G, which was defined in (2.52). On grid interiors,

$$\begin{aligned} G^\ell_{i+\frac{1}{2},j} &= (\frac{\phi^\ell_{i+1,j} - \phi^\ell_{i,j}}{\Delta x^\ell}, \frac{\phi^\ell_{i+1,j+1} + \phi^\ell_{i-1,j+1} - \phi^\ell_{i+1,j-1} - \phi^\ell_{i-1,j-1}}{4 \Delta y^\ell})^T \\ G^\ell_{i,j+\frac{1}{2}} &= (\frac{\phi^\ell_{i+1,j+1} + \phi^\ell_{i+1,j-1} - \phi^\ell_{i-1,j+1} - \phi^\ell_{i-1,j-1}}{4 \Delta x^\ell}, \frac{\phi^\ell_{i,j+1} - \phi^\ell_{i,j}}{\Delta y^\ell})^T. \end{aligned} \tag{4.16}$$

Where this stencil crosses a coarse-fine interface with the coarser level $\ell-1$, we will use the quadratic interpolation operator I from Section 3.1.2 to fill ghost cells around the level ℓ grid, which will then be used in the normal stencil for G^ℓ.

Then, definition of the cell-centered level-operator gradient $G^{CC,\ell}$ is straightforward, using the edge-to-cell averaging operator $Av^{E\to C}$:

$$G^{CC,\ell}\phi = Av^{E\to C}G^\ell\phi. \tag{4.17}$$

Note that the G^ℓ operator contains the coarse-fine boundary conditions for $G^{CC,\ell}$, since the edge-centered gradient is defined on coarse-fine interfaces with coarser levels through the coarse-fine interpolation operator I. For this reason, it will not be necessary to explicitly define a coarse-fine boundary condition for the cell-centered gradient operator.

Divergence

Similar to the level-operator gradient, we will first define the level-operator version of the edge-centered divergence operator, D^ℓ. We can then use (2.62) to define the cell-centered level-operator divergence D^{CC}.

Recall that the edge-centered divergence operator D is a cell-centered divergence of edge-centered quantities. We define the level-operator D^ℓ of the edge-centered vector field $\mathbf{u} = (u,v)^T$ in the same way:

$$D^\ell\mathbf{u} = \frac{u^\ell_{i+\frac{1}{2},j} - u^\ell_{i-\frac{1}{2},j}}{\Delta x^\ell} + \frac{v^\ell_{i,j+\frac{1}{2}} - v^\ell_{i,j-\frac{1}{2}}}{\Delta y}. \tag{4.18}$$

Since the edge which makes up the coarse-fine interface with the coarser level $\ell-1$ is considered to be a part of the level ℓ, there is no need to specify a coarse-fine boundary condition for this operator.

To define the cell-centered divergence operator $D^{CC,\ell}$, we will once again draw on the definition of the cell-centered divergence operator in (2.62):

$$D^{CC,\ell}\mathbf{u} = D^{\ell}(Av^{C\rightarrow E}\mathbf{u}^{\ell}). \tag{4.19}$$

where $Av^{C\rightarrow E}$ is the cell-to-edge averaging operator.

So, the boundary condition for the level-operator cell-centered divergence $D^{CC,\ell}$ where the stencil crosses a coarse-fine interface with level $\ell - 1$ is defined by the boundary conditions set for $\mathbf{u}^{\ell}$ before averaging to edges. Examination of the level-advance algorithm in the previous section shows that in most cases, the divergence operator will be applied to the intermediate velocity field $\mathbf{u}^{**,\ell}$ to compute the right-hand-side for the level projection. Because of the subcycled nature of the level ℓ timestep, it is not clear what, if any, coarse-grid quantity would be appropriate to use as a coarse-grid boundary condition for $\mathbf{u}^{**,\ell}$ (for example, $\mathbf{u}^{**,\ell-1}$ has the wrong centering in time). For this reason, it was decided to use linear extrapolation of $\mathbf{u}^{\ell}$ to compute ghost-cell values for $\mathbf{u}^{\ell}$ at coarse-fine interfaces.

Advective Terms

We also must compute advective terms in the level update, both the $[(\mathbf{u}\cdot\nabla)\mathbf{u}]^{\ell}$ terms in the momentum equation and the $F^{S,\ell} = \nabla\cdot(\mathbf{u}s)$ term in the advection update, as well as in the computation of the advection velocities. This is similar to the method described in Section 2.6.1, with the addition of suitable coarse-fine boundary conditions. This part of the level advance has a hyperbolic character to it, and is similar to the hyperbolic conservation laws solved in Section 4.1. This consists of extrapolating values for $\mathbf{u}$ and s to edges at time $(t^{\ell} + \frac{\Delta t^{\ell}}{2})$, then using the upwinded values to compute the advective updates. Because of the hyperbolic nature of this part of the update, we use the same coarse-fine boundary conditions that were used in 4.1, which was conservative interpolation of coarse solution values in time and space. Before the tracing step, coarse-grid solution values are interpolated in time to t^ℓ, and then are conservatively interpolated in space to fill ghost cells around each grid. Once this is done, the single-grid tracing and upwinding algorithm of Section 2.6.1 can be used in a straightforward manner.

The computation of advection velocities includes a level-operator version of the edge-centered projection described in Section 2.6.1. We first solve:

$$L^\ell \phi^\ell = D^\ell(\mathbf{u}^{n+\frac{1}{2},\ell}), \tag{4.20}$$

and then correct the velocity field to make it divergence-free:

$$\mathbf{u}^{AD,\ell} = \mathbf{u}^{n+\frac{1}{2},\ell} - G^\ell \phi^\ell \tag{4.21}$$

Note that we have not specified the coarse-fine boundary condition for ϕ^ℓ in (4.20) and (4.21); we will defer this issue until the specification of the entire adaptive algorithm in Section 4.5.

4.3 A Simple Recursive Timestep

Once the level advance algorithm of the last section has been defined, it is straightforward to extend the methodology of Berger and Colella to the incompressible Euler equations. To ensure proper coupling between levels, the appropriate velocity and scalar flux registers $\delta\mathbf{V}$ and δs are maintained, and coarse grid velocities and scalars are corrected to ensure conservation by the refluxing operation described in Section 4.1.2. The pseudocode for the recursive timestep for this algorithm is shown in Figure 4.6. Unfortunately, this algorithm suffers from two significant problems. Both of these issues were identified by Almgren et al. [5] in the construction of their adaptive projection method for the incompressible Navier-Stokes equations.

First, the composite velocity field will not satisfy the divergence constraint based on composite operators. We would expect that since the divergence constraint was enforced using level

```
EulerAdvance($\ell, t^\ell, \Delta t^\ell$)
    FillGhostCells($\ell, t^\ell$)
    Compute $\mathbf{u}^{AD,\ell}$
    Compute advective fluxes: $\mathbf{F}^{S,\ell}$
    $s^\ell_{i,j}(t^\ell + \Delta t^\ell) := s^\ell_{i,j}(t^\ell) - \frac{\Delta t^\ell}{\Delta x^\ell}\left(F^{S,\ell}_{x,i+\frac{1}{2},j} - F^{S,\ell}_{x,i+\frac{1}{2},j}\right) - \frac{\Delta t^\ell}{\Delta y^\ell}\left(F^{S,\ell}_{y,i,j+\frac{1}{2}} - F^{S,\ell}_{y,i,j-\frac{1}{2}}\right)$
    Update scalar flux registers:
        if ($\ell < \ell_{max}$) $\delta s^{\ell+1} = -\mathbf{F}^{\mathbf{S},\ell^\ell} \cdot \mathbf{n}^{\ell+1}_{CF}$    on $\partial\Omega^{\ell+1}$
        if ($\ell > 0$) $\delta s^\ell = \delta s^\ell + \frac{1}{n^{\ell-1}_{ref}}\langle F^{S,\ell} \cdot \mathbf{n}^\ell_{CF}\rangle$    on $\partial\Omega^\ell$
    Compute velocity advection $[(\mathbf{u} \cdot \nabla)\mathbf{u}]^\ell$
    Update velocity flux registers
        if ($\ell < \ell_{max}$) $\delta \mathbf{V}^{\ell+1} = -(\mathbf{u}^{AD,\ell} \cdot \mathbf{n}^{\ell+1}_{CF})\mathbf{u}^{half,\ell}$    on $\partial\Omega^{\ell+1}$
        if ($\ell > 0$) $\delta \mathbf{V}^\ell = \delta \mathbf{V}^\ell + \frac{1}{n^{\ell-1}_{ref}}\langle -(\mathbf{u}^{AD,\ell} \cdot \mathbf{n}^{\ell+1}_{CF})\mathbf{u}^{half,\ell}\rangle$    on $\partial\Omega^\ell$
    $\mathbf{u}^{**,\ell} := \mathbf{u}^\ell(t^\ell) - \Delta t^\ell[(\mathbf{u} \cdot \nabla)\mathbf{u}]^\ell$
    Solve $L^\ell \pi^\ell = D^{CC,\ell}\mathbf{u}^{**,\ell}$
    $\mathbf{u}^\ell(t^\ell + \Delta t^\ell) := \mathbf{u}^{**,\ell} - \Delta t^\ell G^{CC,\ell}\pi^\ell$
    if ($\ell < \ell_{max}$) then
        for $n = 0, n^\ell_{ref} - 1$
            $\Delta t^{\ell+1} = \frac{1}{n^\ell_{ref}}\Delta t^\ell$
            $t^{\ell+1} = t^\ell + n\Delta t^{\ell+1}$
            EulerAdvance($\ell + 1, t^{\ell+1}, \Delta t^{\ell+1}$)
        end for
        AvgDown($s^\ell(t^\ell + \Delta t^\ell), s^{\ell+1}(t^\ell + \Delta t^\ell)$)
        AvgDown($\mathbf{u}^\ell(t^\ell + \Delta t^\ell), \mathbf{u}^{\ell+1}(t^\ell + \Delta t^\ell)$)
        Reflux: $s^\ell(t^\ell + \Delta t^\ell) = s^\ell(t^\ell + \Delta t^\ell) - \Delta t^\ell D_R(\delta s^\ell)$
        Reflux: $\mathbf{u}^\ell(t^\ell + \Delta t^\ell) = \mathbf{u}^\ell(t^\ell + \Delta t^\ell) - \Delta t^\ell D_R(\delta \mathbf{V}^\ell)$
    end if
end EulerAdvance
```

Figure 4.6: Naive extension of Berger-Colella algorithm to incompressible Euler

operators in a level-operator based level projection, that the composite pressure field π computed by the level projections will not satisfy the elliptic matching condition described in Section 3.2.1. For example, if the initial velocity field is divergence-free, and $(\mathbf{u} \cdot \nabla)\mathbf{u}$ is independent of time, enforcing the divergence constraint using level projections corresponds to Dirichlet-only matching for the pressure solve. An equivalent statement is that there is no sense in which the jump in the normal velocities $[\mathbf{u} \cdot \mathbf{n}_{CF}]$ is zero at coarse-fine interface. Another issue is that velocity refluxing has modified the coarse-level velocity fields in a row of cells one-cell wide around coarse-fine interfaces with the finer level. This velocity was not included in the coarse-level level projection, and so will cause a violation of the divergence constraint.

Also, this scheme will not be freestream preserving. Although we project the advection velocities $\mathbf{u}^{AD}$ with an edge-centered projection (Section 2.6), the projection we use is also based on level operators, and so the correction field ϕ also does not satisfy the elliptic matching condition. This means that while the advection velocities are divergence-free based on a level-operator discretization, they are not generally divergence-free based on composite divergence operators.

As a result, errors in advection will occur at coarse-fine interfaces. While our advection scheme will be conservative due to refluxing, it will not be freestream preserving. The resulting errors will be apparent in the evolution of a scalar field which is initially constant throughout the domain. Because of the non-solenoidal nature of the advection velocity field, this scalar, which should maintain its constant value throughout its evolution, will begin to show errors at coarse-fine interfaces, as it sees the effects of local contractions and expansions of the non-solenoidal advection velocity field at the coarse-fine interfaces.

For example, consider the two-level case. Assume that a scalar s has a constant value s_0

in a region surrounding a coarse-fine interface. In this case, the coarse-grid update will produce the correct solution, because the fluxes based on the coarse-grid solution will balance and lead to no net change in s. The flux register for this coarse-fine interface will contain

$$\begin{aligned} \delta s^{\ell+1} &= -s_0 \mathbf{u}^{AD,\ell} \cdot \mathbf{n}_{CF} + \frac{1}{n_{ref}} \sum \langle s_0 \mathbf{u}^{AD,\ell+1} \cdot \mathbf{n}_{CF} \rangle \\ &= s_0 \big(-\mathbf{u}^{AD,\ell} \cdot \mathbf{n}_{CF} + \frac{1}{n_{ref}} \sum \langle \mathbf{u}^{AD,\ell+1} \cdot \mathbf{n}_{CF} \rangle \big), \end{aligned} \tag{4.22}$$

where $\mathbf{u}^{AD}$ is the advection velocity, the summation is over the subcycled fine-level timesteps, and the $\langle\rangle$ denotes an arithmetic average of the fine-level edge-centered values on the coarse-fine interface. Since the coarse-grid update has already produced the correct solution, we would like the refluxing correction to have no effect. For that to happen, (4.22) implies that the coarse-grid advection velocities $\mathbf{u}^{AD,\ell} \cdot \mathbf{n}_{CF}$ must equal the average of the fine-grid advection velocities $\frac{1}{n_{ref}} \sum \langle \mathbf{u}^{AD,\ell+1} \cdot \mathbf{n}_{CF} \rangle$. However, because the coarse- and fine-grid advection velocities were computed using independent Taylor extrapolations and single-level elliptic solves, there is no guarantee this will be the case. As a result, we expect that the refluxing operation, while it preserves conservation of s, will generate violations of freestream preservation, and s will not equal s_0 in the cells immediately adjacent to the coarse-fine interface. Once these errors have been made, they will then be advected throughout the flow, contaminating the solution in regions away from coarse-fine interfaces.

In this work, we address these issues by constructing a multilevel projection which is applied at the end of each coarse timestep, after the refluxing operations have been performed. This will ensure that the composite velocity field satisfies a composite divergence constraint. Also, we will introduce a supplementary advected quantity to track freestream-preservation errors. We can then use this quantity to compute corrections to the advective velocity field to make the scheme approximately freestream preserving. In [5], these issues are resolved using somewhat different techniques;

a comparison of the two algorithms will be deferred until Section 4.6.1.

4.4 Additions to Hyperbolic Algorithm for Incompressible Flow

As described in the previous section, a naive extension of the algorithm of Berger and Colella to the incompressible Euler equations suffers from two serious weaknesses, both springing from the fact that the divergence constraint has been applied on a level-by-level basis, rather than in a composite sense. In this section, we describe the steps taken in this work to fix these problems.

4.4.1 Composite Projection

We would like our composite velocity field $\mathbf{u}$ to satisfy the divergence constraint (2.40) based on a composite divergence operator, rather than one based on the level divergence operator used in the level projections.

After the subcycled level solves, the resulting composite velocity field will not, in general, satisfy the divergence constraint based on composite operators, even though we performed level projections on the velocity field during each level solve. This is the same effect seen in Section 3.2.1. While we used pressure information from coarser levels as a boundary condition for the finer grids, this represents only a Dirichlet boundary condition for the pressure – Neumann matching has not been enforced. Once again, the solution on the finer levels has seen the effect of the coarser grids, but the coarse-level pressure field has not seen the effect of the finer levels.

In addition, the refluxing operation for velocity has altered the coarse-level velocity field, adding a set of velocities to a ring of coarse cells one cell wide around the projection of the fine grids. This added velocity field was never projected at all, and so a correction must be made to ensure that the refluxed velocities do not cause the composite velocity field to violate the divergence constraint.

To correct for these problems, we will define a composite projection, which will be based on composite operators and will be applied to the composite multilevel velocity field. This projection will be applied during the synchronization step, after the refluxing operations have been performed; for this reason, we will call this multilevel projection the *synchronization projection*

While we have already defined the cell-centered Laplacian operator we will use in this work in Section 3.1.2, we will need to define composite analogs of the single-level cell-centered D^{CC} and G^{CC} operators defined in Section 2.5.2. We expect that they will be similar to the cell-centered level operators defined in Section 4.2.2 (which already contain coarse-fine boundary conditions with coarser levels), with the addition of coarse-fine boundary conditions in the form of matching conditions with a finer level, if it exists. As in Section 3.1.2, we also expect that away from coarse-fine interfaces, the composite operator discretizations will reduce to the appropriate single-level cell-centered discretization. We also expect that definition of the cell-centered operators D^{CC} and G^{CC} as edge-centered operators with appropriate cell-to-edge and edge-to-cell averaging (equations (2.62) and (2.63)) will prove useful, since we have already defined composite MAC-centered divergence and gradient operators in Section 3.1.2.

Composite Divergence

In Section 3.1.2, we defined a composite edge-centered divergence operator. Our composite cell-centered divergence operator will be similar. Away from the coarse-fine interface, as usual, the divergence will be the normal cell-centered $D^{CC,\ell}$ operator of (2.62). So, on the fine grid, away from the coarse-fine interface,

$$D^{CC,comp}\mathbf{u} = D^{CC,fine}\mathbf{u}^{fine}. \tag{4.23}$$

On the coarse grid away from the interface,

$$D^{CC,comp}\mathbf{u} = D^{CC,crse}\mathbf{u}^{crse}. \tag{4.24}$$

We will once again need to define a special composite operator wherever the stencil of the normal coarse or fine divergence crosses a coarse-fine interface. As seen in (2.62), the cell-centered divergence operator can be defined as an edge-centered divergence (2.51) of edge-centered velocities created by averaging cell-centered velocities to edges. At coarse-fine interfaces, we will compute the fine-level edge-centered velocities by using linear extrapolation to compute cell-centered velocities in ghost cells surrounding the fine grid, and then using these values in the standard $Av^{C\rightarrow E}$ operator of (2.61). We will then define the appropriate coarse-level edge-centered velocity on the coarse-fine interface edge as the arithmetic average of the edge-centered velocities used to compute the divergences on the fine side of the coarse-fine interface.

So, on the fine side of the coarse-fine interface,

$$\begin{aligned} D^{CC,comp,CF}\mathbf{u} &= D^{fine}(\mathbf{u}^{edge,fine}) \\ \mathbf{u}^{edge,fine} &= Av^{C\rightarrow E}\mathbf{u}^{fine} \end{aligned} \tag{4.25}$$

where ghost-cell values for $\mathbf{u}^{fine}$ along the coarse-fine interface are computed using linear extrapolation of the interior values of $\mathbf{u}^{fine}$, for consistency with the level-operator divergence operator. On the coarse side of the coarse-fine interface,

$$\begin{aligned} D^{CC,comp,CF}\mathbf{u} &= D^{crse}(\mathbf{u}^{edge,crse}) \\ \mathbf{u}^{edge,crse} &= \begin{cases} \langle \mathbf{u}^{edge,fine} \rangle & \text{on } \partial\Omega^{fine} \\ Av^{C\rightarrow E}\mathbf{u}^{crse} & \text{elsewhere.} \end{cases} \end{aligned} \tag{4.26}$$

Composite Gradient

We defined a composite edge-centered gradient operator in Section 3.1.2 when we defined the "fluxes" in the composite Laplacian operator. We will use this definition, along with (2.63), to construct our composite cell-centered gradient operator. Since most of the gradients we will be computing will be of quantities like pressure, which are defined by solving an elliptic equation, this is appropriate. However, we will need to alter the operator on the coarse side of the interface due to the structure of many of the fields to which we will apply the gradient operator.

We will use the edge-centered composite gradient G^{comp} defined in Section 3.1.2, where quadratic coarse-fine interpolation, along with flux-matching, were used to define the gradient operator at coarse-fine interfaces. We reiterate here for completeness.

Away from coarse-fine interfaces, the composite gradient is simply the coarse- or fine-level edge-centered gradient of (2.52):

$$\begin{aligned} G^{comp}\phi^{comp} &= \begin{cases} G^{crse}\phi^{crse} & \text{on } \Omega^{crse} \\ G^{fine}\phi^{fine} & \text{on } \Omega^{fine} \end{cases} \\ \phi^{fine} &= I(\phi^{fine},\phi^{crse}) \quad \text{on } \partial\Omega^{\ell} \end{aligned} \tag{4.27}$$

On the coarse-fine interface, we define the fine edge-centered gradients by using the quadratic coarse-fine interpolation operator from Section 3.1.2 to define ghost-cell values for ϕ along the coarse-fine interface, and then using the normal fine-level G operator to compute the edge-centered gradients. The coarse-level values for the gradient along the coarse-fine interface will be defined as the arithmetic average of the fine-level gradients which overlie the coarse edge.

We can then average this edge-centered composite gradient to cell centers to define a cell-

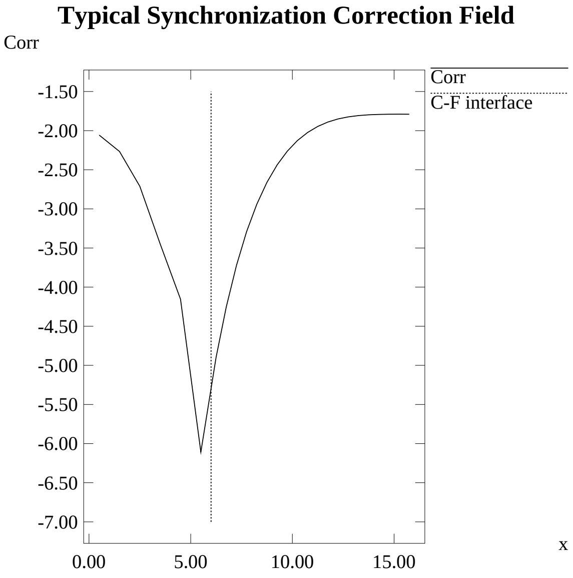


Figure 4.7: Typical synchronization correction, corr. Fine grid is to the right of the coarse-fine interface.

centered composite gradient operator $G^{CC,comp}$, as in (2.63), which is repeated here for convenience:

$$G^{CC,comp}\phi = Av^{E\rightarrow C}G^{comp}\phi, \tag{4.28}$$

In practice, the only place we will actually apply the composite cell-centered gradient operator will be during synchronization operations. While the discretization of the synchronization projection will be discussed in the next section, the structure of the resulting correction fields necessitated a modification to the definition of the gradient operator. In most cases, the source terms for the synchronization projection are primarily in a set of cells one cell wide on the coarse side of the coarse-fine interface, in essence a $\delta-$function in the direction normal to the interface. A $\delta-$function source distribution to Poisson's equation implies a solution which, although continuous, has a discontinuity in the first derivative (See Figure 4.7). In this case, computing the edge-centered gradients using G^{comp} and then averaging to produce a cell-centered gradient will wash out the structure of the gradient field near the interface because the strongly positive and negative gradients on either side of the discontinuity will cancel. The solution to this problem is to compute the derivative in a one-sided way from the coarse side of the coarse-fine interface and use this one-sided gradient for the coarse cell immediately adjacent to the coarse-fine interface. This preserves the structure of the correction across the coarse-fine interface.

So, in regions of the fine grid away from the coarse-fine interface, we compute the cell-centered gradient fields according to (2.63):

$$G^{CC,comp}\phi^{comp} = G^{CC,fine}\phi^{fine} \quad \text{on} \quad \Omega^{\text{fine}}. \tag{4.29}$$

On the coarse grid away from the fine grid, we likewise use the standard coarse discretization:

$$G^{CC,comp}\phi^{comp} = G^{CC,crse}\phi^{crse} \quad \text{on} \quad \Omega^{\text{crse}}. \tag{4.30}$$

On the fine side of the coarse-fine interface, we first use the quadratic interpolation operator (Section 3.1.2) to compute cell-centered fine-grid ghost-cell values. Then, following (2.63), we compute edge-centered gradients, which are then averaged to cell-centers:

$$\begin{aligned} G^{CC,comp}\phi^{comp} &= G^{CC,fine}\phi^{fine} \\ \phi^{fine} &= I(\phi^{fine}, \phi^{crse}) \quad \text{on} \quad \partial\Omega^{\text{fine}} \end{aligned} \tag{4.31}$$

On the coarse side of the coarse-fine interface, we will use linear extrapolation of the edge-centered gradients to provide an edge-centered gradient on the coarse-fine interface. For example, if the coarse-fine interface is located at the $(i-\frac{1}{2}, j)$ edge (see Figure 4.8), then we compute the cell-centered gradient at coarse cell (i, j) as follows:

$$\begin{aligned} (G^{edge,crse}\phi)_{i-\frac{1}{2},j} &= 2(G^{edge,crse}\phi)_{i+\frac{1}{2},j} - (G^{edge,crse}\phi)_{i+\frac{3}{2},j} \\ (G^{CC,comp}\phi)_{i,j} &= Av^{E\rightarrow C}(G^{edge,crse}\phi). \end{aligned} \tag{4.32}$$

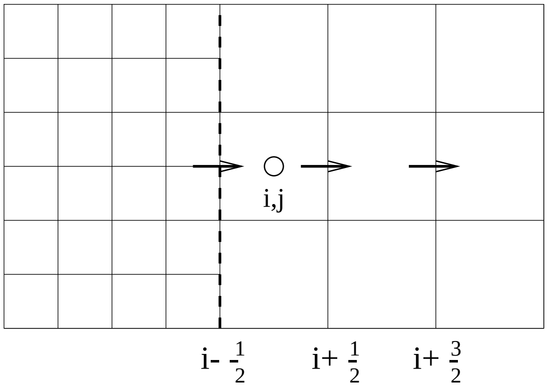


Figure 4.8: Computing the composite gradient on the coarse side of a coarse-fine interface for cell (i, j), when coarse-fine interface is located at $(i - \frac{1}{2}, j)$ edge. Edge-centered gradient at $(i - \frac{1}{2}, j)$ is computed by linear extrapolation of edge-centered gradients at $(i + \frac{1}{2}, j)$ and $(i + \frac{3}{2}, j)$. Edge-centered gradients at $(i + \frac{1}{2}, j)$ and $(i - \frac{1}{2}, j)$ are averaged to cell center to get $G^{comp}\phi$ at (i, j)

Away from the coarse-fine interface, $G^{edge,crse}\phi$ will be the edge-centered gradient of (2.52).

Discretization of Composite Projection

During the synchronization step, we will perform a synchronization projection to ensure that the velocity field satisfies the composite divergence constraint. If we separate the pressure field into the contribution from the level projections and the remaining correction,

$$p = \pi + e_s, \tag{4.33}$$

then constructing the synchronization projection becomes straightforward. Since the correction due to π has already been included in the velocities, we now use e_s to enforce the composite constraint by first solving:

$$\begin{aligned} L^{comp} e_s &= \frac{1}{\Delta t^{sync}} D^{CC,comp} \mathbf{u}(t^{sync}) \\ e_s^{\ell_{base}} &= I(e_s^{\ell_{base}}, e_s^{\ell_{base}-1}) \end{aligned} \tag{4.34}$$

with appropriate physical boundary conditions (if necessary), and then correcting the velocity field:

$$\begin{aligned} \mathbf{u}^{new} &= \mathbf{u}^{new} - \Delta t^{sync} G^{CC,comp} e_s \qquad (4.35) \\ e_s^{\ell_{base}} &= I(e_s^{\ell_{base}}, e_s^{\ell_{base}-1}), \end{aligned}$$

where $G^{CC,comp}$ is the one-sided composite gradient operator defined in Section 4.4.1, and Δt^{sync} is the timestep of the coarsest level which is at t^{sync}, in essence the timestep over which the synchronization is being applied. The appropriate physical boundary conditions for e_s will be the homogeneous form of the boundary condition applied to the level pressure π. For solid walls, this will be a homogeneous Neumann boundary condition.

The projection discretization we are using is approximate, in the sense that $D^{CC,comp} G^{CC,comp} \neq L^{comp}$, where L^{comp} is the 5-point Laplacian operator, $L^{comp} = D^{comp} G^{comp}$ (D^{comp} and G^{comp} are the edge-centered divergence and gradient operators). In this case, the discrete projection operator is:

$$P = I - Av^{E \rightarrow C} G^{comp} (L^{comp})^{-1} D^{comp} Av^{C \rightarrow E}. \qquad (4.36)$$

The use of the averaging operators are what make this projection approximate. Because the discretization of the projection operator used in this work is approximate, the projection is not idempotent; in other words, $P^2 \neq P$.

We would like to show that repeated application of the composite projection will be well behaved, in that it will be stable, and that the resulting velocity field will converge to to a consistent solution. For a uniform grid with periodic boundary conditions, Lai [44] showed using Fourier analysis that this projection discretization is stable, in that $||P|| \leq 1$, and that repeated application of the projection will drive the divergence to zero, or

$$D^{CC,comp}(P^N \mathbf{u}) \rightarrow 0$$

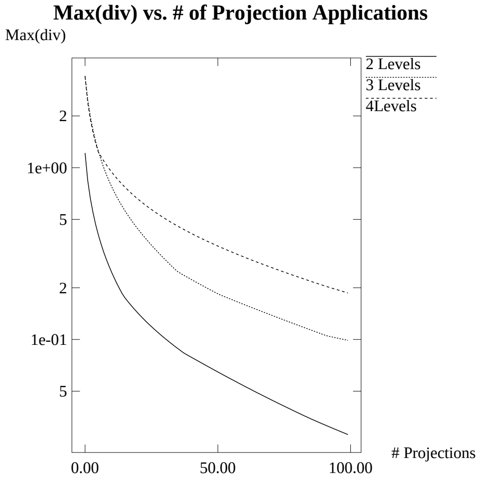


Figure 4.9: Max(divergence) vs. number of repeated projection applications

as $N \to \infty$, where $P^N(\mathbf{u})$ represents the repeated application of the projection N times.

To demonstrate the effectiveness and stability of this composite projection, we repeatedly applied the composite projection to a sample problem and evaluated the results. This was performed on the three-vortex test case described in Section 6.1.3. Figure 4.9 shows $Max(D^{comp}\mathbf{u})$ against the number of times the projection was applied. It can be seen that the composite divergence does go to zero as the projection is repeatedly applied. Adding more levels of refinement affects the rate that the divergence is decreased, but does not appear to affect the general behavior. Also, we would expect that each new projection has a smaller effect on the solution, as the velocity field converges toward one which is completely divergence-free. The amount that each application of the projection changes the solution (in other words, $P^n - P^{n+1}$) is equal to $(I-P)P^n$, which from (2.46) is just the gradient piece, $G^{CC,comp}e$. Figure 4.10 shows $max(G^{CC,comp}(e))$, which is the maximum that the solution is changed in a given application of the projection. As can be seen, this quantity decreases monotonically as the projection is repeatedly applied. The magnitude of the correction is

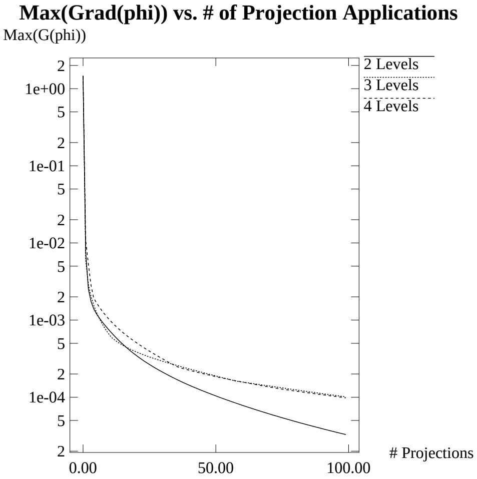


Figure 4.10: Max of the gradient piece returned by the projection vs. number of repeated projection applications.

much larger in the first application of the projection because that is where the physical boundary condition (solid walls, in this case) is being enforced; the velocity field is initialized as if it were in infinite space, and then the initial projection also enforces the physical boundary conditions.

4.4.2 Freestream Preservation

To correct errors in freestream preservation, we follow the volume-discrepancy approach used by Propp [51], which in turn is based on work by Acs et al. [1], and Trangenstein and Bell [66].

We start with a scalar field initialized to one everywhere in the domain, which we shall call Λ. As we advance the solution, we also compute advective updates to Λ, using (2.66) and following the algorithm detailed for passive scalars in Sections 2.6.2 for the single-grid case, and which will be described in 4.5.2 for the multilevel case. Since we know that Λ should remain one, $\Lambda \neq 1$ is a good indicator of the advection errors that are being made.

We will compute a correction to the advection velocities which will return Λ to one, undoing the errors which have been made. Since we are correcting for errors in the advection velocities, we cast the correction as a velocity field $\mathbf{u}_p$ which we add to the advection velocities. We would like our correction velocity field to undo freestream errors which are manifest by $\Lambda \neq 1$. From the advection equation (2.66),

$$D(\mathbf{u}\Lambda) = \frac{\Lambda^n - \Lambda^{n+1}}{\Delta t}. \tag{4.37}$$

Since we would like to return Λ to 1, we set Λ^{n+1} to one. Also, we assume that the errors in Λ are small, and so we treat the Λ inside the divergence as constant and pull it outside the divergence:

$$D(\mathbf{u}_p + \mathbf{u}^{AD}) = \frac{\Lambda^n - 1}{\Lambda \Delta t}. \tag{4.38}$$

Since $\mathbf{u}^{AD}$ is essentially divergence-free, $D\mathbf{u}^{AD} \approx 0$, and we are left with the correction field. If we define the correction field $\mathbf{u}_P$ as a gradient,

$$\mathbf{u}_p = Ge_\Lambda, \tag{4.39}$$

then we are left with an elliptic equation to solve:

$$Le_\Lambda = \frac{\Lambda^n - 1}{\Lambda \Delta t}, \tag{4.40}$$

where L is the Laplacian operator. Similar to the projection operator, the physical boundary conditions for e_Λ are

$$\nabla e_\Lambda = \mathbf{u}_p \cdot \mathbf{n}, \tag{4.41}$$

which in the case of solid walls reduces to a homogeneous Neumann boundary condition on e_Λ.

Solving (4.40) for e_Λ, we can then compute the correction velocity field $\mathbf{u}_p = G^{comp}e_\Lambda$, which we then add to the advection velocity field in future timesteps. Since this will be done as a synchronization operation, we will take Δt to be Δt^{sync}. Note that both $\mathbf{u}_p$ and $\mathbf{u}^{AD}$ have edge-centering. The correction velocity field $\mathbf{u}_p$ will tend to correct the errors made in advection and will work to drive Λ back to one.

In practice, we make two modifications to (4.40). First, we assume that the Λ in the denominator of the right hand side is approximately one. In that case, the right hand side becomes $\frac{\Lambda-1}{\Delta t}$. This change is made to ensure that the elliptic equation is solvable. Also, we include a scaling term, η, to adjust the strength of the correction. So, the equation we solve during the synchronization step is:

$$\begin{aligned} Le_\Lambda &= \frac{\Lambda - 1}{\Delta t^{sync}}\eta \\ e_\Lambda^{\ell_{base}} &= I(e_\Lambda^{\ell_{base}}, e_\Lambda^{\ell_{base}-1}) \quad \text{on} \quad \partial\Omega^{\ell_{\text{base}}}. \end{aligned} \tag{4.42}$$

Note that we have explicitly included the coarse-fine boundary condition for the case where $\ell_{base} > 0$. In this usage, the parameter η has a meaning – it is the reciprocal of the number of ℓ_{base} timesteps it will take for Λ to return to one. We have found that values for η which are greater than one are unstable, because they introduce an overcorrection. If we express the modified evolution of Λ using (4.42), we find that it is a forward-Euler update of the equation

$$\frac{D\Lambda}{Dt} = \frac{\eta}{\Delta t}(\Lambda - 1). \tag{4.43}$$

For values of η greater than one, the forward Euler scheme we are using is unstable. So, $\eta \leq 1$ for stability. We have found that $\eta = 0.9$ has worked well for the problems examined in this work.

When the gradient field $\mathbf{u}_p$ is added to the advective velocity field $\mathbf{u}^{AD}$, then $\mathbf{u}^{AD}$ is no longer divergence-free, even in the grid interior regions. For this reason, we must use convective differencing when computing the advective terms of the velocity update, rather than a conservative discretization. This is why the single-grid algorithm outlined in Section 2.6 employs convective differencing when computing the advective terms in Section 2.6.3.

4.5 Complete Multilevel Algorithm

The following sections will describe the recursive timestep used to advance the solution on level ℓ from time t^{ℓ} to $t^{\ell} + \Delta t^{\ell}$. A basic pseudocode outline of our recursive level update is shown in Figure 4.11. Like the recursive timestep for hyperbolic conservation laws described in Section 4.1.3, the level ℓ timestep implicitly includes the subcycled advance of all finer levels and synchronization with those levels, producing a composite solution for levels finer than and including level ℓ.

The synchronization strategy will be somewhat different, however. In Section 4.1.3, levels are synchronized in coarse-fine pairs. For example, in a three-level solution, at the end of a level 0 timestep, they first synchronize levels 1 and 2, and then synchronize levels 0 and 1. In this work, we will perform synchronization operations which involve elliptic solves over all levels which have reached the same time, which we will call t^{sync}. We do this because of the results of Section 3.3, in which it was shown that structuring a multilevel elliptic solution as a series of level solves and then making corrections to coarse-fine pairs of levels is less accurate than performing a single multilevel solve. This means that the elliptic solves used in the composite projection and in the freestream preservation algorithm will be performed for all levels which have reached t^{sync}. This is done by testing to see if the $\ell - 1$ level has reached the time t^{sync} before performing a synchronization with the current level ℓ as ℓ_{base}.

We will now describe each step in the algorithm in turn.

```
EulerLevelAdvance($\ell, t^\ell, \Delta t^\ell$)
    Compute Advection Velocities $\mathbf{u}^{AD,\ell}$
    Compute Advective Fluxes $\mathbf{F}^{\mathbf{S},\ell}, \mathbf{F}^{\mathbf{\Lambda},\ell}$
    Compute Advective Updates:
        $s^\ell_{i,j}(t^\ell + \Delta t^\ell) := s^\ell_{i,j}(t^\ell) - \frac{\Delta t^\ell}{\Delta x^\ell}\left(F^{S,\ell}_{x,i+\frac{1}{2},j} - F^{S,\ell}_{x,i-\frac{1}{2},j}\right) - \frac{\Delta t^\ell}{\Delta y^\ell}\left(F^{S,\ell}_{y,i,j+\frac{1}{2}} - F^{S,\ell}_{y,i,j-\frac{1}{2}}\right)$
        $\Lambda^\ell_{i,j}(t^\ell + \Delta t^\ell) := \Lambda^\ell_{i,j}(t^\ell) - \frac{\Delta t^\ell}{\Delta x^\ell}\left(F^{\Lambda,\ell}_{x,i+\frac{1}{2},j} - F^{\Lambda,\ell}_{x,i-\frac{1}{2},j}\right) - \frac{\Delta t^\ell}{\Delta y^\ell}\left(F^{\Lambda,\ell}_{y,i,j+\frac{1}{2}} - F^{\Lambda,\ell}_{y,i,j-\frac{1}{2}}\right)$
    Predict $\mathbf{u}^{half}$
    $\mathbf{u}^{**,\ell}_{i,j} = \mathbf{u}^\ell_{i,j}(t^\ell) - \Delta t[(\mathbf{u}\cdot\nabla)\mathbf{u}]^{n+\frac{1}{2}}_{i,j}$
    Update advective and velocity Flux Registers:
        if ($\ell < \ell_{max}$) then
            $\delta s^{\ell+1} = -\mathbf{F}^{\mathbf{S},\ell}\cdot\mathbf{n}^{\ell+1}_{CF}$   on $\partial\Omega^{\ell+1}$
            $\delta\Lambda^{\ell+1} = -\mathbf{F}^{\mathbf{\Lambda},\ell}\cdot\mathbf{n}^{\ell+1}_{CF}$   on $\partial\Omega^{\ell+1}$
            $\delta\mathbf{V}^{\ell+1} = -(\mathbf{u}^{AD,\ell}\cdot\mathbf{n}^{\ell+1}_{CF})\mathbf{u}^{half,\ell}$   on $\partial\Omega^{\ell+1}$
        end if
        if ($\ell > 0$) then
            $\delta s^\ell = \delta s^\ell + \frac{1}{n^{\ell-1}_{ref}}\langle F^{S,\ell}\cdot\mathbf{n}^\ell_{CF}\rangle$   on $\partial\Omega^\ell$
            $\delta\Lambda^\ell = \delta\Lambda^\ell + \frac{1}{n^{\ell-1}_{ref}}\langle F^{\Lambda,\ell}\cdot\mathbf{n}^\ell_{CF}\rangle$   on $\partial\Omega^\ell$
            $\delta\mathbf{V}^\ell = \delta\mathbf{V}^\ell + \frac{1}{n^{\ell-1}_{ref}}\langle(\mathbf{u}^{AD,\ell}\cdot\mathbf{n}^\ell_{CF})\mathbf{u}^{half,\ell}\rangle$   on $\partial\Omega^\ell$
        end if
    Project $\mathbf{u}^{**,\ell} \rightarrow \mathbf{u}^\ell(t^\ell + \Delta t^\ell)$ :
        Solve $L^\ell\pi^\ell = \frac{1}{\Delta t^\ell}D^{CC,\ell}\mathbf{u}^{**,\ell}$
        $\mathbf{u}^\ell(t^\ell + \Delta t^\ell) = \mathbf{u}^{**,\ell} - \Delta t^\ell G^{CC,\ell}\pi^\ell$
    if ($\ell < \ell_{max}$)
        $\Delta t^{\ell+1} = \frac{1}{n^\ell_{ref}}\Delta t^\ell$
        for $n = 0, n^\ell_{ref} - 1$
            EulerLevelAdvance($\ell + 1, t^\ell + n\Delta t^{\ell+1}, \Delta t^{\ell+1}$)
        end for
        if ($t^\ell + \Delta t^\ell < t^{\ell-1} + \Delta t^{\ell-1}$)   Synchronize($\ell, t^\ell + \Delta t^\ell, t^\ell$)
    end if
end EulerLevelAdvance
```

Figure 4.11: Recursive level timestep for the incompressible Euler equations.

4.5.1 Computing Advection Velocities

First, we need to compute advection velocities with which to compute advective updates for scalars and velocities. This will be similar to the single-level algorithm in Section 2.6.1. First, we must fill ghost cells around each grid on this level. Coarse-fine boundary conditions are computed by conservative linear interpolation in space and time of the velocities on the coarser level. Once again, because we have already advanced the level $\ell - 1$ solution to time $t^{\ell-1} + \Delta t^{\ell-1}$, we will be able to interpolate the old and new coarse-level solutions $\mathbf{u}^{\ell-1}(t^{\ell-1})$ and $\mathbf{u}^{\ell-1}(t^{\ell-1} + \Delta t^{\ell-1})$ in time to t^{ℓ}. Then $\mathbf{u}^{\ell-1}(t^{\ell})$ is spatially interpolated using conservative linear interpolation to fill the ghost cells around the level ℓ grids. Due to the stencils involved in the predictor step, it is necessary to fill a ring of ghost cells more than one cell thick to have all the necessary information for this step. Once again, ghost cells in zones where level ℓ grids abut each other are filled by copying $\mathbf{u}^{\ell}(t^{\ell})$ solution values from the interiors of other level ℓ grids, and physical boundary conditions are set in the same way as for the single-grid problem.

Once the ghost cells have been filled, we then predict edge-centered velocities $\mathbf{u}^{n+\frac{1}{2}}$ in exactly the same way as was done in Section 2.6.1. We first use a Taylor extrapolation to predict left and right (top and bottom for the $y-$direction) edge-centered values at time $t^{\ell} + \frac{\Delta t^{\ell}}{2}$, and then choose the upwind state at each edge.

Then we perform an edge-centered projection on these predicted velocities to ensure that the advection velocities are divergence-free. This is also a straightforward extension of the single-grid edge-centered projection described in Section 2.6.1. We first compute the edge-centered divergence of $\mathbf{u}^{n+\frac{1}{2},\ell}$ using the operator D^{ℓ}. Note that there are no explicit coarse-fine boundary conditions necessary for this operator, because we have predicted edge-centered velocities along the boundary with level $\ell - 1$ using the interpolated coarse-level velocities. We then solve

$$L^\ell \phi^\ell = D^\ell \mathbf{u}^{n+\frac{1}{2},\ell} \tag{4.44}$$

$$\phi^\ell = I(\phi^\ell, \frac{\Delta t}{2}[\pi^{\ell-1} + e_s^{\ell-1}]) \quad \text{on } \partial\Omega^\ell. \tag{4.45}$$

The coarse-fine boundary condition on ϕ^ℓ is designed to ensure matching with the total pressure field, which is $\pi + e_s$. We solve for ϕ^ℓ using the level solver algorithm outlined in Section 3.2.6.

We then correct the edge-centered advection velocities as in Section 2.6.1, using the level-operator version of the edge-centered gradient:

$$\mathbf{u}^{AD,\ell} = \mathbf{u}^{n+\frac{1}{2},\ell} - G^\ell \phi^\ell \tag{4.46}$$

$$\phi^\ell = I(\phi^\ell, \frac{\Delta t^\ell}{2}[\pi^{n-\frac{1}{2},\ell-1} + e_s^{\ell-1}]).$$

Finally, we include the effects of the freestream preservation correction from Section 4.4.2:

$$\mathbf{u}^{AD,\ell} = \mathbf{u}^{AD,\ell} + \mathbf{u}_p \tag{4.47}$$

where $\mathbf{u}_p = G^{comp} e_\Lambda$.

4.5.2 Scalar Advection

Once we have the advection velocities $\mathbf{u}^{AD,\ell}$, we can compute the updated scalar fields, $s(t^\ell + \Delta t^\ell)$. As in the velocity predictor, the scalar predictor will use interpolated coarse-level boundary conditions for s^ℓ, interpolated in time and space using conservative interpolation. Once the boundary conditions have been set, the scalar update follows the algorithm outlined in Section 2.6.2. First, we compute edge-centered upwinded values for $s^{n+\frac{1}{2},\ell}$, and then use these to compute the fluxes, which we use to perform the scalar update:

$$s^\ell(t^\ell + \Delta t^\ell) = s^\ell(t^\ell) - \Delta t D^\ell(\mathbf{u}^{AD,\ell} s^{n+\frac{1}{2},\ell}) \tag{4.48}$$

Anticipating the refluxing correction which will be performed later, we initialize and/or update flux registers as necessary with the fluxes. If a finer level exists, we initialize the level $\ell + 1$ flux registers with the coarse-level fluxes, and if $\ell > 0$, we update the level ℓ flux registers with fine fluxes as detailed in Figure 4.11. The vector $\mathbf{n}^{\ell}_{\mathbf{CF}}$ is the local normal of the coarse-fine interface for level ℓ.

We also advance Λ^{ℓ}, the freestream preservation indicator, in the same way as the advected scalars s^{ℓ}. The level $\ell/\ell + 1$ coarse-fine mismatch information for Λ is stored in the flux registers $\delta\Lambda^{\ell+1}$, which are analogous to $\delta s^{\ell+1}$.

4.5.3 Velocity Predictor

As in the single-grid algorithm, we now compute the advective component of the velocity update. Using the advection velocities $\mathbf{u}^{AD,\ell}$, we now predict the tangential components of the edge-velocities $\mathbf{u}^{half,\ell}$ as in Section 2.6.3. As before, we use conservative linear interpolation in time and space from the coarse-level data to fill a ring of ghost cells around fine-grids for use in the prediction step. Also, as in Section 2.6.3, we must now include the effects of $G^{\ell}\phi^{\ell}$ in these predicted velocities.

The computation of advection velocities in the multilevel algorithm differs from the single level algorithm in the addition of $\mathbf{u}^{\ell}_{p}$ to correct for coarse-fine errors. In the single-grid algorithm, we use the edge-centered $\mathbf{u}^{AD,\ell}$ as the edge-centered $\mathbf{u}^{half,\ell}$ normal to the cell edges. In the adaptive algorithm, we must first remove the effects of $\mathbf{u}_p$ from $\mathbf{u}^{AD,\ell}$:

$$\begin{aligned} u^{half,\ell}_{i+\frac{1}{2},j} &= u^{AD,\ell}_{i+\frac{1}{2},j} - u_{p,i+\frac{1}{2},j} \\ v^{half,\ell}_{i,j+\frac{1}{2}} &= v^{AD,\ell}_{i+\frac{1}{2},j} - v_{p,i,j+\frac{1}{2}}. \end{aligned} \tag{4.49}$$

1. Predict edge-centered $\mathbf{u}^{half,\ell}$
2. $\mathbf{u}^{**,\ell} = \mathbf{u}^{n,\ell} - \Delta t^{\ell}[Av^{E\rightarrow C}(\mathbf{u}^{AD,\ell}) \cdot G^{\ell}\mathbf{u}^{half,\ell}]$
3. Update velocity flux registers:
    - If $\ell < \ell_{max}$, $\quad \delta\mathbf{V}^{\ell+1} = -(\mathbf{u}^{AD,\ell} \cdot \mathbf{n}_{CF}^{\ell+1})\mathbf{u}^{half,\ell} \quad \text{on} \quad \partial\Omega^{\ell+1}$
    - If $\ell > 0$, $\quad \delta\mathbf{V}^{\ell} = \delta\mathbf{V}^{\ell} + \frac{1}{n_{ref}^{\ell-1}}\langle(\mathbf{u}^{AD,\ell} \cdot \mathbf{n}_{CF}^{\ell+1})\mathbf{u}^{half,\ell}\rangle \quad \text{on} \quad \partial\Omega^{\ell}$

Figure 4.12: Velocity predictor portion of level advance algorithm

In essence, we are discriminating between the *advecting* velocity field $\mathbf{u}^{AD,\ell}$ and the *advected* velocity field $\mathbf{u}^{half,\ell}$.

Once the edge-centered velocities have been computed, we compute the advective terms $[(\mathbf{u} \cdot \nabla)\mathbf{u}]^{n+\frac{1}{2},\ell}$ using equations (2.87). Note that $\mathbf{u}^{AD,\ell}$ contains the effects of the freestream preservation correction $\mathbf{u}_p$. Since we have computed all necessary edge velocities, there are no explicit coarse-fine boundary conditions necessary for this step. The intermediate velocity $\mathbf{u}^{**,\ell}$ can now be computed, using (2.88).

As in the scalar update, we now anticipate the velocity refluxing in the synchronization step by initializing and/or updating velocity flux registers. If a finer level exists, we initialize its velocity flux register with the velocity fluxes across the coarse-fine interface, $(\mathbf{u}^{AD,\ell} \cdot \mathbf{n}_{CF}^{\ell+1})\mathbf{u}^{half,\ell}$. If a coarser level exists, we increment it with the average of the velocity fluxes across the interface. Note that we will be refluxing both normal and tangential components of velocity, so in two dimensions, the flux register $\delta\mathbf{V}$ has two components. See Figure 4.12.

4.5.4 Level Projection

Once $\mathbf{u}^{**,\ell}$ has been computed, all that remains in the level advance is to perform the level projection, which will approximately enforce the divergence constraint using level operators. Similar to the single-level projection in Section 2.6.4, we solve:

$$\begin{aligned} L^\ell \pi^\ell(t^\ell + \frac{1}{2}\Delta t^\ell) &= \frac{1}{\Delta t^\ell} D^{CC,\ell}\mathbf{u}^{**,\ell} \\ \pi^\ell(t^\ell + \frac{1}{2}\Delta t^\ell) &= I(\pi^\ell(t^\ell + \frac{1}{2}\Delta t^\ell), \pi^{\ell-1}(t^\ell + \frac{1}{2}\Delta t^\ell)) \quad \text{on} \quad \partial\Omega^\ell \end{aligned} \tag{4.50}$$

where $\pi^{\ell-1}(t^\ell + \frac{1}{2}\Delta t^\ell)$ denotes linear interpolation or extrapolation of $\pi^{\ell-1}$ in time using the old and new coarse pressures $\pi^{\ell-1}(t^{\ell-1} - \frac{1}{2}\Delta t^{\ell-1})$ and $\pi^{\ell-1}(t^{\ell-1} + \frac{1}{2}\Delta t^{\ell-1})$. Equation (4.50) is solved using the level solver algorithm described in Section 3.2.6. The velocity on the current level is then corrected with the gradient of π^ℓ:

$$\begin{aligned} \mathbf{u}^\ell(t^\ell + \Delta t^\ell) &= \mathbf{u}^{**} - \Delta t G^{CC,\ell}\pi^\ell(t^\ell + \frac{1}{2}\Delta t^\ell) \\ \pi^\ell(t^\ell + \frac{1}{2}\Delta t^\ell) &= I(\pi^\ell(t^\ell + \frac{1}{2}\Delta t^\ell), \pi^{\ell-1}(t^\ell + \frac{1}{2}\Delta t^\ell)) \quad \text{on} \quad \partial\Omega^\ell. \end{aligned} \tag{4.51}$$

4.5.5 Subcycled Advance of Finer Levels

If a finer level $\ell+1$ exists, it is now advanced n^ℓ_{ref} times with $\Delta t^{\ell+1} = \frac{1}{n^\ell_{ref}}\Delta t^\ell$. Implicit in the subcycled advances of level $\ell+1$ are the subcycled advances of all levels finer than $\ell+1$ and any necessary intermediate synchronizations between level $\ell+1$ and finer levels. Once this is complete, all levels finer than level ℓ will also be at $t^\ell + \Delta t^\ell$.

4.5.6 Synchronization

At this point, we synchronize level ℓ will all finer levels. As mentioned earlier, we first check to see if a coarser level has also reached the same time as the current level. If this is the case, we

```
Synchronize($\ell_{base}, t^{sync}, \Delta t^{sync}$)
    AverageDown:
        for $\ell = \ell_{max} - 1, \ell_{base}, -1$
            $\mathbf{u}^{\ell}(t^{sync}) = Avg(\mathbf{u}^{\ell+1}(t^{sync}))$   on $\mathrm{P}(\Omega^{\ell+1})$
            $s^{\ell}(t^{sync}) = Avg(s^{\ell+1}(t^{sync}))$   on $\mathrm{P}(\Omega^{\ell+1})$
            $\Lambda^{\ell}(t^{sync}) = Avg(\Lambda^{\ell+1}(t^{sync}))$   on $\mathrm{P}(\Omega^{\ell+1})$
        end for
    Reflux:
        for $\ell = \ell_{max} - 1, \ell_{base}, -1$
            $\mathbf{u}^{\ell}(t^{sync}) := \mathbf{u}^{\ell}(t^{sync}) - \Delta t^{\ell} D_R(\delta \mathbf{V}^{\ell+1})$
            $s^{\ell}(t^{sync}) := s^{\ell}(t^{sync}) - \Delta t^{\ell} D_R(\delta s^{\ell+1})$
            $\Lambda^{\ell}(t^{sync}) := \Lambda^{\ell}(t^{sync}) - \Delta t^{\ell} D_R(\delta \Lambda^{\ell+1})$
        end for
    Synchronization Projection:
        Solve $L^{comp} e_s = \frac{1}{\Delta t^{sync}} D^{CC,comp} \mathbf{u}(t^{sync})$   for $\ell \geq \ell_{\mathrm{base}}$
        $\mathbf{u}(t^{sync}) := \mathbf{u}(t^{sync}) - \Delta t^{sync} G^{CC,comp} e_s$   for $\ell \geq \ell_{\mathrm{base}}$
    Freestream Preservation Solve:
        Solve $L^{comp} e_{\Lambda} = \frac{(\Lambda(t^{sync}-1)}{\Delta t^{sync}} \eta$   for $\ell \geq \ell_{\mathrm{base}}$
        $\mathbf{u}_p = G^{comp} e_{\Lambda}$
end Synchronize
```

Figure 4.13: Synchronization for incompressible Euler equations.

do the synchronization operations for all levels which are at the current time $t^{sync} = t^{\ell} + \Delta t^{\ell}$. If we denote the coarsest level which has reached t^{sync} as ℓ_{base}, we synchronize all levels $\ell \geq \ell_{base}$. The timestep over which the synchronization is being performed is then $\Delta t^{sync} = \Delta t^{\ell_{base}}$. A pseudocode description of the synchronization algorithm is in Figure 4.13.

Averaging Fine Data to Coarser Levels

As in the hyperbolic algorithm in Section 4.1, we first replace the solution on coarse grids which are covered by refinement with averaged fine-level solutions. This is done from the finer levels down to coarser levels, so that the solution in all regions is replaced by the appropriately averaged finest solution possible. This averaging down operation is done for the velocity field $\mathbf{u}(t^{sync})$ the scalar $s(t^{sync})$, and the freestream preservation quantity $\Lambda(t^{sync})$.

Refluxing

To ensure conservation, we then perform a refluxing operation for velocity and the advected scalars s and Λ. This will be similar to the refluxing operations described for the hyperbolic algorithm in Section 4.1, and is essentially a reflux-divergence of the mismatch of the fluxes, which have been stored in the appropriate flux registers. Note that both normal and tangential (to the coarse-fine interface) components of velocity in coarse cells adjacent to coarse-fine interfaces are updated in this step.

Synchronization Projection

To ensure that the composite velocity field satisfies the divergence constraint based on composite operators, the composite projection described in Section 4.4.1 is applied to the composite velocity field for all levels ℓ and finer. We solve:

$$\begin{aligned} L^{comp} e_s^{comp} &= \frac{1}{\Delta t^{sync}} D^{CC,comp} \mathbf{u}^{comp} \\ e_s^{\ell_{base}} &= I(e_s^{\ell_{base}}, e_s^{\ell_{base}-1}) \quad \text{on } \partial\Omega^{\ell_{\mathrm{base}}} \end{aligned} \tag{4.52}$$

for the levels ℓ_{base} and higher, using the multilevel solver algorithm described in Section 3.2.5. We then correct the velocities for levels $\ell_{base} \leq \ell \leq \ell_{max}$:

$$\begin{aligned} \mathbf{u}^{\ell}(t^{sync}) &= \mathbf{u}^{\ell}(t^{sync}) - \Delta t^{sync} G^{CC,comp} e_s \\ e_s^{\ell_{base}} &= I(e_s^{\ell_{base}}, e_s^{\ell_{base}-1}). \end{aligned} \tag{4.53}$$

Freestream Preservation Correction

The last synchronization operation is to compute the freestream preservation correction velocities $\mathbf{u}_p$. This is similar to the synchronization correction in that it involves a multilevel solve for all levels $\ell \geq \ell_{base}$. In this case, we solve (4.42):

$$\begin{aligned} L^{comp} e_{\Lambda} &= \frac{\Lambda - 1}{\Delta t^{sync}} \eta \\ e_{\Lambda}^{\ell_{base}} &= I(e_{\Lambda}^{\ell_{base}}, e_{\Lambda}^{\ell_{base}-1}). \end{aligned} \tag{4.54}$$

Then, the gradient of the correction u_p can be computed and stored for future use:

$$\begin{aligned} \mathbf{u}_p = G^{comp} e_{\Lambda} \quad \text{for } \ell \geq \ell_{\text{base}} \\ e_{\Lambda}^{\ell_{base}} = I(e_{\Lambda}^{\ell_{base}}, e_{\Lambda}^{\ell_{base}-1}). \end{aligned} \tag{4.55}$$

This completes the synchronization operations, which in turn completes the level ℓ_{base} timestep.

4.6 Initialization

Before the initial timestep for a level ℓ, initial values for $\pi^{\ell-1}$ and $e_s^{\ell-1}$ will need to be computed for use as boundary conditions. Also, the initial velocity field must be projected to ensure that it satisfies the composite divergence constraint. Moreover, if a new grid configuration for level

ℓ is defined as a result of regridding operations during the computation, initial values for π^ℓ, e_s^ℓ, and $\mathbf{u}_p$ will need to be computed for the new grids. Note that after regridding, e_s and $\mathbf{u}_p$ will need to be recomputed on the finest unchanged level as well.

As mentioned previously, at the beginning of the computation the initial velocity field must be projected to ensure that it satisfies the composite divergence constraint. This is a straightforward application of the Hodge-Helmholtz decomposition (2.41), extracting the divergence-free component of the velocity field. We solve:

$$L^{comp} e_{init}^{comp} = D^{CC,comp} \mathbf{u}_{init}^{comp} \tag{4.56}$$

over the entire grid hierarchy, using the multilevel algorithm presented in Sections 3.2.3 and 3.2.4. Physical boundary conditions are imposed appropriately on the velocity and the correction field e_{init} as in the single-grid projection, described in Section 2.5.1. Then, the velocity field is corrected onto the space of vectors which satisfies the divergence constraint:

$$\mathbf{u}^{comp} = \mathbf{u}_{init}^{comp} - G^{CC,comp} e_{init}^{comp}. \tag{4.57}$$

As before, appropriate physical boundary conditions are applied, based on the single-level projection boundary conditions.

For initialization purposes, we will define ℓ_{base} as the finest *unchanged* level in the grid hierarchy. For initialization before the initial timestep, ℓ_{base} will be -1. The basic strategy will be to compute a single non-subcycled timestep on all grids $\ell > \ell_{base}$, in the process computing all the required quantities. Because the usual edge-centered projection in the advection step uses e_s as a boundary condition, we compute two iterations of the initialization timestep – one in which e_s is not used as a coarse boundary condition for ϕ, and then a second one where the e_s computed in the first iteration is used for the coarse boundary condition for ϕ. Since the initialization process is not subcycled, the timestep $\widetilde{\Delta t}$ will be dependent on the stability requirements of the finest level. In practice, we use half of the timestep we would normally use on the finest level:

$$\begin{aligned} \widetilde{\Delta t} &= \frac{1}{2}\Delta t^{\ell_{max}} \\ &= \frac{1}{2}\sigma\frac{\Delta x^{\ell_{max}}}{max(\mathbf{u}^{\ell_{max}})}, \end{aligned} \tag{4.58}$$

where σ is the CFL number, defined in (4.5)

The algorithm used to initialize π and e_s is shown in Figure 4.14. If the initialization is being performed after a regridding operation, instead of at the initial step, then there are also advection errors from previous timesteps which much be corrected as well. The advection correction $\mathbf{u}_p$ is based on the current Λ field, rather than on one computed in an initialization timestep, because the goal of the freestream preservation correction is to correct for errors which have already occurred, while the goal of the initialization timestep is to predict reasonable values for π and e_s.

Initializing π

To initialize π, we do a non-subcycled level advance on each level greater than ℓ_{base}. Since the coarse-fine boundary conditions for the edge-centered projection require $e_s^{\ell-1}$, which has not yet been computed, we do two passes of the initialization algorithm. During the first pass, e_s is not available, so we use the coarse-level ϕ as the boundary condition for all levels greater than ℓ_{base}. During the second pass, we can use the estimate for e_s and π computed during the previous timesteps:

$$\begin{aligned} L^\ell\phi^\ell &= D^\ell\tilde{\mathbf{u}}^{half,\ell} \\ \phi^\ell &= \begin{cases} I(\phi^\ell,\phi^{\ell-1}) & \text{if n = 1} \\ I(\phi^\ell,\frac{\widetilde{\Delta t}}{2}(\pi^{\ell-1}+e_s^{\ell-1})) & \text{otherwise} \end{cases} \end{aligned}$$

```
EulerInit($\ell_{base}, t^{init}$)
    Compute $\widetilde{\Delta t}$
    for $n = 1, n_{passes}$
        if ($\ell_{base} > -1$)
            Compute $\tilde{\mathbf{u}}^{**,\ell_{base}}$ as usual
            Project $\tilde{\mathbf{u}}^{**,\ell_{base}}$:
                Solve $L^{\ell_{base}}\tilde{\pi}^{\ell_{base}} = \frac{1}{\widetilde{\Delta t}}D^{CC,\ell_{base}}\tilde{\mathbf{u}}^{**,\ell_{base}}$
                $\tilde{\mathbf{u}}^{\ell_{base}} := \tilde{\mathbf{u}}^{**,\ell_{base}} - \widetilde{\Delta t}G^{CC,\ell_{base}}\tilde{\pi}^{\ell_{base}}$
        end if
        for $\ell = \ell_{base} + 1, \ell_{max}$
            Predict $\tilde{\mathbf{u}}^{half,\ell}$ as in Sect. 4.5.3
            Perform edge-centered projection of advection velocities: $L^{\ell}\phi^{\ell} = D^{\ell}\tilde{\mathbf{u}}^{half,\ell}$
            Correct advection velocities: $\tilde{\mathbf{u}}^{AD,\ell} := \tilde{\mathbf{u}}^{AD,\ell} - G^{\ell}\phi^{\ell}$
            Predict $\tilde{\mathbf{u}}^{half,\ell}$ as in Sect 4.5.3: $\tilde{\mathbf{u}}^{**,\ell} = \mathbf{u}^{\ell} - \Delta t^{\ell}[Av^{E\to C}(\tilde{\mathbf{u}}^{AD,\ell}) \cdot \nabla\tilde{\mathbf{u}}^{half,\ell}]$
            Update velocity flux registers:
                if ($\ell < \ell_{max}$)   $\delta\mathbf{V}^{\ell+1} = -(\tilde{\mathbf{u}}^{AD,\ell} \cdot \mathbf{n}_{CF})\tilde{\mathbf{u}}^{half,\ell}$   on $\partial\Omega^{\ell+1}$
                if ($\ell > 0$)   $\delta\mathbf{V}^{\ell} = \delta\mathbf{V}^{\ell} + \langle(\tilde{\mathbf{u}}^{AD,\ell} \cdot \mathbf{n}_{CF})\tilde{\mathbf{u}}^{half,\ell}\rangle$   on $\partial\Omega^{\ell}$
            Project $\tilde{\mathbf{u}}^{**,\ell} \to \tilde{\mathbf{u}}^{\ell}(t^{\ell} + \Delta t^{\ell})$ :
                Solve $L^{\ell}\pi^{\ell} = \frac{1}{\widetilde{\Delta t}}D^{CC,\ell}\tilde{\mathbf{u}}^{**,\ell}$
                $\tilde{\mathbf{u}}^{\ell}(t^{\ell} + \Delta t^{\ell}) = \tilde{\mathbf{u}}^{**,\ell} - \widetilde{\Delta t}G^{CC,\ell}\pi^{\ell}$
        end for
        for $\ell = \ell_{max} - 1, \ell_{base}, -1$
            Reflux: $\tilde{\mathbf{u}}^{\ell} := \tilde{\mathbf{u}}^{\ell} - \widetilde{\Delta t}D_R(\delta\mathbf{V}^{\ell+1})$
        end for
        Compute initial $e_s$: Solve $L^{comp}e_s = \frac{1}{\widetilde{\Delta t}}D^{CC,comp}\tilde{\mathbf{u}}$   for $\ell \geq \ell_{\text{base}}$
        Compute initial $\tilde{\mathbf{u}}_p$:
            Solve $L^{comp}e_{\Lambda} = \frac{(\Lambda-1)}{\Delta t^{\ell_{base}}}\eta$   for $\ell \geq \ell_{\text{base}}$
            $\mathbf{u}_p = G^{comp}e_{\Lambda}$
    end for
end EulerInit
```

Figure 4.14: Initialization algorithm for the incompressible Euler equations

Once $\widetilde{\mathbf{u}}^{**,\ell}$ has been computed, we project to get the initial estimate of the level pressure π^ℓ. Note that because the initialization timestep is uniformly $\widetilde{\Delta t}$ for all levels, π^ℓ will have a time centering at $t^{init} + \frac{1}{2}\widetilde{\Delta t}$ for all levels. At coarse-fine interfaces with coarser levels, the boundary condition for π will reflect this centering:

$$\pi^\ell(t^{init} + \frac{1}{2}\widetilde{\Delta t}) = I(\pi^\ell(t^{init} + \frac{1}{2}\widetilde{\Delta t}), \pi^{\ell-1}(t^{init} + \frac{1}{2}\widetilde{\Delta t}) \quad \text{on } \partial\Omega^\ell. \tag{4.59}$$

Initializing e_s

To compute an initial e_s, we first require a set of level-projected velocities for all levels $\ell \geq \ell_{base}$. We expect that e_s will change on the coarsest *unchanged* level because it will reflect the coarse-fine interface corrections for the new finer levels. For this, we will need to compute a level-projected velocity $\widetilde{\mathbf{u}}^{\ell_{base}}(t^{init} + \widetilde{\Delta t})$. For this reason, if $\ell_{base} \neq -1$, we perform an initialization timestep for ℓ_{base} as well.

Once level-projected velocities $\widetilde{\mathbf{u}}^\ell(t^{init} + \frac{1}{2}\widetilde{\Delta t})$ have been computed for all levels $\ell \geq \ell_{base}$, we can then compute the initial estimate for the composite pressure correction e_s by performing an initial synchronization projection. We solve:

$$\begin{aligned} L^{comp} e_s &= \frac{1}{\widetilde{\Delta t}} D^{CC,comp} \widetilde{\mathbf{u}} \quad \text{for } \ell \geq \ell_{\text{base}} \\ e_s^{\ell_{base}} &= I(e_s^{\ell_{base}}, e_s^{\ell_{base}-1}), \end{aligned} \tag{4.60}$$

using the same multilevel projection used in the usual synchronization projection.

Initializing Freestream Preservation Correction

Finally, when initializing a new hierarchy of grids after a regridding operation, we will need to compute a new correction field for freestream preservation errors. Unlike the initializations for

π and e_s, we use the existing solution for Λ at t^{init}, since the goal in this case is to correct for the errors which have already occurred. So, the synchronization timestep in this case will be the timestep of the ℓ_{base}. As in a normal synchronization procedure, we first solve for e_Λ:

$$\begin{aligned} L^{comp} e_\Lambda &= \frac{\Lambda - 1}{\Delta t^{\ell_{base}}} \eta \quad \text{for } \ell \geq \ell_{\text{base}} \\ e_\Lambda^{\ell_{base}} &= I(e_\Lambda^{\ell_{base}}, e_\Lambda^{\ell_{base}-1}). \end{aligned} \tag{4.61}$$

Then, we define the correction velocity field which will be added to the advection velocities:

$$\begin{aligned} \mathbf{u}_p &= G^{comp} e_\Lambda \quad \text{for } \ell \geq \ell_{\text{base}} \\ e_\Lambda^{\ell_{base}} &= I(e_\Lambda^{\ell_{base}}, e_\Lambda^{\ell_{base}-1}). \end{aligned} \tag{4.62}$$

4.6.1 Comparison to Previous Work

Various approaches have been used to compute adaptive solutions to incompressible flows. To compute steady-state solutions to the incompressible Navier-Stokes equations, Thompson and Ferziger [65] used an adaptive multigrid method based on the adaptive multigrid algorithm originally developed by Brandt [24].

For time-dependent incompressible flows, Howell and Bell [41] constructed an adaptive projection method based on the exact projection and the projection formulation of Bell, Colella, and Glaz[16], in which there was no refinement in time. It was noted that the decoupled stencil of the exact projection caused considerable complications at coarse-fine interfaces because the decoupling of the computational grids had to be respected across coarse-fine interfaces.

Minion [48] constructed a non-subcycled adaptive version of the approximate cell-centered projection of Lai [44] and the projection formulation of [17], which included a multilevel edge-centered projection for advection velocities.

In atmospheric modeling, the anelastic equations for atmospheric motions are similar in structure to those for incompressible flow, with a divergence constraint on velocity which includes the effect of atmospheric stratification. Clark and Farley [30] and Stevens [59, 60] have constructed adaptive methods for anelastic atmospheric dynamics based on the projection method which were fully adaptive in both time and space. Neither of these methods, however, enforced the divergence constraint in a composite sense across all levels of refinement, instead enforcing it on a level-by-level basis, with boundary conditions interpolated from coarser grids. The lack of coupling of the coarse-level pressures to the fine levels has been shown [60, 5] to cause a loss of accuracy in the final solution.

Almgren et al. [5] have developed an adaptive projection method which refines in time as well as space and which enforces the divergence constraint in a composite sense across all levels. Their projection operator is based on the nodal scheme of Almgren, Bell, and Szymczak [4], and uses the basic projection formulation of [16] as extended by Bell, Colella, and Howell [17]. In contrast with this work, the algorithm of [5] projects the approximation to $\frac{\partial \mathbf{u}}{\partial t}$, rather than the entire intermediate velocity field $\mathbf{u}^{**}$.

In [5], the timestep is structured in a similar way to this work, as a series of recursive updates starting with the coarsest level and then using suitably interpolated coarse level values to construct boundary conditions for the fine-grid updates. Because of temporal refinement, each fine level solution is updated multiple times for each coarse level update. Any time the solutions on two levels of refinement reach the same location in time, they are synchronized.

In [5], elliptic matching of the pressure field is also enforced by means of a synchronization projection which ensures that the composite velocity field satisfies the constraint based on a composite operator. Because $\frac{\partial \mathbf{u}}{\partial t}$ is being projected, the construction of the synchronization projection is somewhat different; the mismatch in $\frac{\partial \mathbf{u}}{\partial t}$ is stored and then used, along with the change to the coarse-grid velocity field caused by velocity refluxing, to explicitly compute the source for the synchronization, which appears as a source term on the coarse level. Recall that in the algorithm presented in this work, the composite velocity field is simply re-projected using composite operators to ensure that elliptic matching is enforced. Moreover, in [5], levels are synchronized in coarse-fine pairs, starting at the finest level and continuing with successively coarser level $\ell/\ell-1$ pairs until all levels at t^{sync} have been synchronized.

Freestream preservation, the property that constant fields of advected quantities in incompressible flow remain constant, is enforced in [5] by a second "MAC synchronization" step, which ensures that the advection velocities also satisfy a divergence constraint based on composite operators. A second advection step is then performed on the coarse level using the correction velocities, and advective corrections are then interpolated to finer levels.

A third difference between the algorithm of [5] and the one presented in this work is in the initialization after re-regridding. Since the pressure in [5] is stored as a composite pressure, rather than separate level-based and correction fields, the existing pressure field is interpolated to provide an existing pressure for newly refined regions. Since we maintain separate level pressure π^ℓ and correction e fields, we must compute a new correction field e (as well as a new freestream-preservation correction $\mathbf{u}_p$) after regridding to account for the new grid configurations.

4.7 Filters

As mentioned in Section 2.7, many researchers have found velocity filtering to be necessary to prevent spurious velocity modes from contaminating the solutions. Some attempt was made to extend the filters used in single-grid projection algorithms to this implementation, but they were unsuccessful. Two strategies were employed. First, an attempt was made to design a filter with composite operators. When this proved unsuccessful, the filters described in [53] were employed on the interiors of grids. Because the goal of filtering is to remove oscillatory modes that the interior discretization of the approximate projection leaves behind, it was felt that simply applying filters on the interiors of grids would be sufficient to reduce these modes, without upsetting the matching conditions at the coarse-fine interfaces.

In practice, however, employing filters in this way caused noticeable vorticity generation at coarse-fine interfaces. In light of this, it was decided to not use filtering at the present time. Other projection method implementations have also not found filtering to be necessary, including that of [27], for example. Almgren et al. only use filtering when necessary to prevent obvious degradation of the solution in the form of "checkerboarding" of velocity. [6] For the test problems computed in this work, we have not found serious degradation of the solutions, so implementation of filtering for this algorithm has been deferred.

Chapter 5

Error Estimation

As can be imagined, the effectiveness of adaptive methods depends strongly on appropriate placement of refinement. Almgren et al. [5] showed that refined patches placed without care did not, in general, reap the benefits of increased resolution. We would like our error estimation criteria to be able to predict where refinement should be placed to improve the accuracy of the solution.

There have been many different approaches to deciding where to place refined patches, ranging from fairly involved mathematical estimates of the error (for example, [19, 67], to fairly simple usage of flow quantities of interest, such as vorticity, density gradients, or energy (for example, [5]). In this work, we have used variations on several of these methods.

In this chapter, we will develop the methods used to estimate where regridding is needed; at present, we use four ways to decide where to place refined patches:

- user-defined grids.
- user-defined criteria
- Richardson extrapolation

The first is not truly an adaptive refinement technique, since the grids are pre-defined. The remaining techniques are automatic, in that they require little or no user input (other than some sort of tolerance for the criteria) and are adaptive, in that they are able to respond to features in the solution as they develop.

For the adaptive grid generation techniques, we follow a two-step process. First, we look at the existing hierarchy, apply our criteria to the current solution to "tag" cells in the existing grid hierarchy for (further) refinement (or un-refinement: if a currently refined cell no longer needs refinement, it is not tagged, and so is no longer included in the list of cells to be refined). Then, we use a clustering algorithm to group these tagged cells into new grids.

5.1 User-Defined Grids

The first, and most straightforward, method of determining grid placement is to use pre-defined grids. This is most useful when the user has a good idea already where refinement will be most beneficial or where there is an already known feature or region of interest in the solution. However, because the grid structures are defined without direct interaction with the solution, this is not really an "adaptive" method per se. As such, it is generally not as useful as fully automated grid generation.

5.2 User-Defined Criteria

There are many cases where the user will have an idea of which features are of interest or are indicative of a need for refinement. To support this, cells can be tagged based on any user-defined solution-based criteria. For example, one may want to tag on areas of high vorticity or high heat release as indicators of interesting features in the solution. It is common to use derivatives of solution quantities like velocity or density as indicators of areas of high activity which could benefit from refinement. In many cases, this is sufficient to improve the quality of the solution, especially if the quantity of interest also has a strong solution-based indicator of the necessity for increased resolution.

While this is a fully automatic and adaptive grid generation technique, it does not necessarily ensure increased accuracy of the solution. Because it is not really an estimation of error, there is no guarantee that refinements based on solution features will improve the solution quality. There is always the chance that important features of the solution will be missed. On the other hand, refinement may be based on spurious features of the solution. Baker [11] raises the possibility that errors due to the grid interfaces can then further excite these spurious features, resulting in further solution degradation. Also, Sweby and Yee [64] demonstrated that refinement based on solution featured can cause chaotic behavior in the case of moving-grid refinement.

In fact, even refinement based on solution error will not necessarily improve solution quality. In many cases, solution errors are nonlocal in nature, resulting from accumulation of discretization errors elsewhere in the domain [11]. Minion [49] also shows that prevention of spurious vortices in incompressible flow can require refinement in locations other than the neighborhood of the spurious feature itself. For this reason, we believe that error estimates based on localized measures of discretization error are a better indicator of where refinement should be placed for greatest benefit.

5.3 Richardson Extrapolation

The use of Richardson Extrapolation to estimate the truncation error of a numerical solution has a long and rich history. Berger and Oliger [19], Berger and Colella [18], and Propp [51] have used it for time-dependent problems, while Berger and Jameson [21], Dudek [34], and Bettencourt [22] have used it for steady-state problems. Variants of this procedure are also used in [21] and [45]. The implementation of Richardson extrapolation in this work is based on that described in Martin and Cartwright [47] and extended by Propp [51].

The basic idea is to apply the operator L to the existing solution, coarsen the result, and then compare it to the operator applied to a coarsened version of the solution. It can be shown that the difference between the two is proportional to the local truncation error. In terms of the existing operators from Chapter 3:

$$Error^{\ell} = Average(L^{\ell}U^{\ell}) - L^{\ell-1}Average(U^{\ell}) \tag{5.1}$$

For steady-state problems, we are generally solving an equation of the form $L(U) = f$. For Poisson's problem, L is the Laplacian operator. For time-dependent problems, the equation we are solving is $\frac{\partial U}{\partial t} = L(U)$, so L is the right-hand-side of the discrete time evolution equation.

As mentioned before, we expect that our scheme will lose accuracy at coarse-fine interfaces and that the local truncation error will be $O(h)$ (one order less accurate than the rest of the scheme) due to the coarse-fine interpolation error. For the same reasons, the scheme will also lose accuracy at physical boundaries, since we are using a lower order approximation there as well. So, if we naively use the error computed using (5.1) there, we will see a large error, which will appear in a single layer of cells on both the coarse and fine sides of the interfaces. Both in theory and in practice, however, this error on the coarse-fine interfaces and physical boundaries does not affect the global accuracy

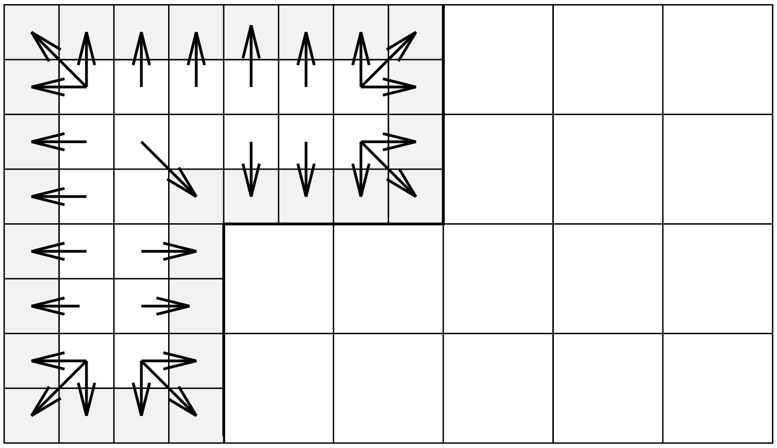


Figure 5.1: Replacing error in fine-grid boundary cells (shaded) with adjacent values

of the scheme, since it is on a set of one dimension less than the problem space. (This assumes, of course, that the surface/volume ratio is of order $\frac{1}{h}$, which will not be true for very small grids, where the surface/volume ratio approaches one.)

So, we do not want to use the error computed by (5.1) on these cells; if we did, refined grids would simply expand until they reached the physical boundaries. We do not, however, want to simply ignore the possibility that we may want to refine these boundary cells. So, for each boundary cell, we copy the error computed on an adjacent cell which is untainted by the coarse-fine boundary error. For the elliptic equations in [47], areas of high error tend to be in patches, rather than single cells; for the Euler equations, we have noticed similar behavior. Since we are dealing with patches of high error, copying from adjacent cells is an adequate solution. On the fine side of the coarse-fine interface, adjacent cell values are copied as shown in Figure 5.1. In [47], cells on the coarse side of the coarse-fine interface were replaced with averages of the adjacent fine-grid values (which had themselves been replaced as necessary). In the case of the Euler equations, the differences in time-centering between solutions on each level preclude this; instead, we simply copy the adjacent coarse values.

5.3.1 Richardson Extrapolation for the Poisson Problem

For the Poisson problem, the error equation (2.14) indicates that reducing the truncation error should certainly result in a reduction of the solution error. For this reason, we expect that truncation error is an excellent indicator of where refinement is necessary to improve solution quality. Moreover, using local truncation error as an indicator will localize the sources of error. Due to the elliptic nature of Poisson's equation, local discretization errors will induce solution errors which are nonlocal in nature; using an estimate of local truncation error to decide where to place refinements will make it possible to localize the sources of the global solution errors.

Since we want to estimate the error on all existing levels, including those partially overlain by refined grids, we need to modify this procedure slightly. Where a grid is covered by a refined patch, we use the error computed on the refined level. Since we know that the error is proportional to h^2, we can rescale the fine error by $(\frac{h_c}{h_f})^2$ (the square of the refinement ratio), and average it onto the coarser grid. This gives a reasonable approximation of what the error in a refined region would be if there were no refinement.

Also, since for Poisson's problem there is no time centering of the different solution levels, we can use the approach in [47] to replace the error computed in coarse cells adjacent to coarse-fine interfaces. In this case, we replace the tainted coarse-cell values with the averaged adjacent fine-grid values, as shown in Figure 5.2.

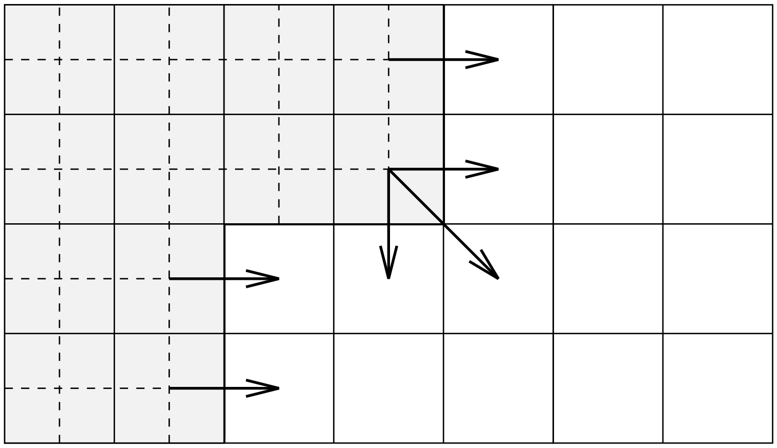


Figure 5.2: Replacing error on the coarse side of the coarse-fine interface with averaged fine-grid values (shaded).

5.3.2 Richardson Extrapolation for the Time-Dependent Problem

For time-dependent problems, the operator L is the discrete time evolution of the solution. We would like the error estimate we construct to have a consistent time centering, so we will take pseudo time steps as depicted in Figure 5.3. The basic strategy will be to do a fine timestep of Δt^{fine} centered around the current time t^n, and then take a coarse time step of $\Delta t^{crse} = 2\Delta t^{fine}$ on the coarsened level, also centered around time t^n. We will then compare the fine and coarse approximations of $\frac{\partial \mathbf{u}}{\partial t}$ to get an estimate of the truncation error. Note that we preserve the time centering of the timestep and also maintain a consistent CFL number for each pseudo-timestep. Note also that this method requires an old solution at $t^n - \Delta t^f$, which we use to compute the initial state for both the coarse and fine approximations; the initial fine state is computed by averaging the t^n and t^{n-1} solutions in time, while the initial coarse state is computed by spatial averaging of the t^{n-1} solution. At the initial timestep, this earlier time does not exist, and so we do not use this method, but instead use a user-defined method to construct the initial grid hierarchy.

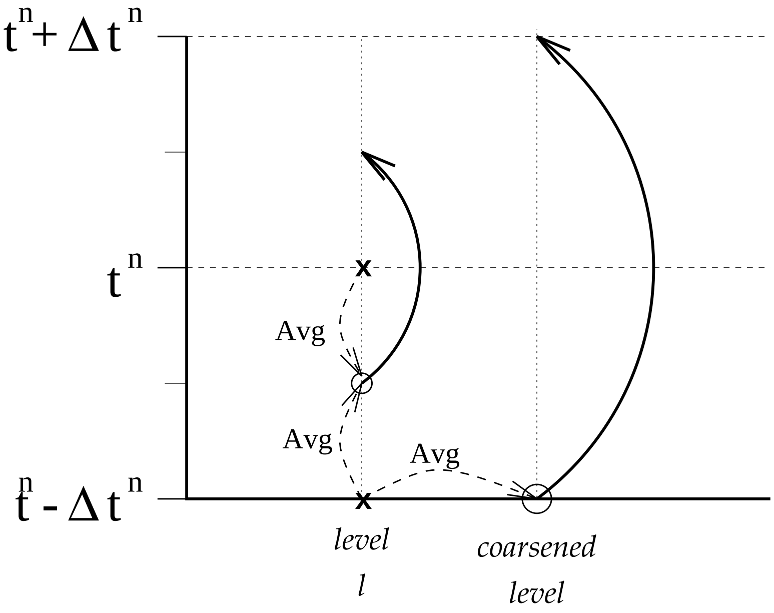


Figure 5.3: Richardson Extrapolation time steps

For the Euler equations, we will also only use the advective part of the complete timestep, $Ad \approx (\mathbf{u} \cdot \nabla)\mathbf{u}$ to compute estimations of the error. This will measure the error in the advection process. It has the advantage of having no nonlocal contributions from the elliptic projection operator, so it is a good localized error measure. Also, since the advection step will provide the source term for the projection, we expect that the errors in advection will be nonlocalized by the projection operator.

We construct fine and coarse approximations as follows:

- Fine approximation

  1. Construct $\mathbf{u}^{n-\frac{1}{2},fine} = \frac{1}{2}(\mathbf{u}^n + \mathbf{u}^{n-1})$
  2. $\Delta t^{fine} = (t^n - t^{n-1})$

3. Predict $\mathbf{u}_G^{half,fine} = (u_{i+\frac{1}{2},j}^{fine,t=n}, v_{i,j+\frac{1}{2}}^{fine,t=n})$ as in Section 4.5.3
4. MAC Projection: $L\phi^{fine} = \nabla \cdot \mathbf{u}_G^{half,fine}$ as in Section 4.5.1
5. Correct predicted velocities: $\mathbf{u}_{ad}^{fine} = \mathbf{u}_G^{half} - G\phi^{fine}$ as in Section 4.5.1
6. Store $G\phi^{fine}$ for future use on coarsened level.
7. Re-predict velocities: trace $\mathbf{u}^{half,fine}$ as in Section 4.5.3
8. $Ad^{fine} = -D(\mathbf{u}_{ad}^{fine} \cdot \mathbf{u}^{half,fine})$
9. $F_{vel}^{fine} = -\mathbf{u}_{ad}^{fine} \cdot \mathbf{u}^{half,fine}$

- Coarse Approximation

  1. Construct $\mathbf{u}^{n-1,crse} = Avg^{fine \rightarrow crse}(\mathbf{u}^{n-1,\ell})$ and

     $G\phi^{crse} = Avg^{fine \rightarrow crse} G\phi^{fine}$ from fine approximation
  2. $\Delta t^{crse} = 2\Delta t^{fine}$
  3. Predict $\mathbf{u}_G^{half,crse} = (u_{i+\frac{1}{2},j}^{crse,t=n}, v_{i,j+\frac{1}{2}}^{crse,t=n})$ as in fine step
  4. Correct predicted velocities: $\mathbf{u}_{ad}^{crse} = \mathbf{u}_G^{half,crse} - G\phi^{crse}$
  5. Re-predict velocities: trace $\mathbf{u}^{half,crse}$
  6. $Ad^{crse} = -D(\mathbf{u}_{ad}^{crse} \cdot \mathbf{u}^{half,crse})$
  7. $F_{vel}^{crse} = -\mathbf{u}_{ad}^{crse} \cdot \mathbf{u}^{half,crse}$

Note that we save an elliptic solve by averaging the MAC projection gradients from the fine approximation for use during the coarse approximation.

We then compute the approximation to the error:

$$E_{AD} = Av^{fine \rightarrow crse}(Ad^{fine}) - Ad^{crse} \tag{5.2}$$

The error we compute in (5.2) has units of $\frac{[L]}{[T]^2}$. We would like to nondimensionalize this so that we can compare it against a nondimensional tolerance. Following the example of the nondimensionalization of the Euler equations in fluid dynamics (see, for example, Schlichting [56]), we use $E^* = \frac{[U]^2}{[L]}$, where $[U]$ is a characteristic velocity (in our case Max($\mathbf{u}$)), and $[L]$ is a characteristic length, which will usually be the length of the problem domain. Then, we can tag on all cells in which the scaled error is greater than a tolerance τ:

$$\frac{E_{AD}}{E^*} > \tau. \tag{5.3}$$

Note E^ℓ_{AD} is actually defined on a grid which is a factor of 2 coarser than Ω^ℓ. This means that when we tag on a cell because it satisfies the criteria in (5.3), we are actually tagging the four (in two dimensions) level ℓ cells which overlie the coarsened grid on which E^ℓ_{AD} is defined.

5.4 Grid Generation

Once we have tagged cells for refinement using one or more of the methods described above, we then must generate suitable block-structured refined grids. This generation process has two conflicting goals. First, we would like to generate efficient grids, in which unnecessary refinement is kept to minimum. This is quantified by defining a grid efficiency,

$$\eta = \frac{\text{number of tagged cells}}{\text{total number of cells refined}} \tag{5.4}$$

and demanding that the grids we generate exceed a prescribed efficiency. We have found an efficiency of around 70-80% to be useful. On the other hand, we would like to generate "block-like" grids which minimize the surface/volume ratio because coarse-fine interfaces carry with them a cost both in computational work needed to enforce synchronization between levels, and because of the error

1. Tag cells using error estimators: T^{ℓ}_{ij} = TRUE if tagged for refinement
2. Coarsen list of tagged cells: $T^{crse} = coarsen(T^{\ell}, F_B)$
3. Call clustering algorithm $\Omega^{crse}_{new} = Cluster(T^{c}rse, \eta_{grids})$
4. Refine grids to new level $\Omega^{\ell+1}_{new} = Refine(\Omega^{crse}_{new}, F_B \times n^{\ell}_{ref})$

Figure 5.4: Basic grid generation algorithm

induced by reduced accuracy at the coarse-fine interface. As was seen in Section 3.4.2, if grids with a high surface/volume ratio are produced, the increased accuracy of the refined patch can be outweighed by the errors induced at the coarse-fine interfaces. Also, because we will be using multigrid acceleration for our elliptic solvers, we would like to have grid configurations which are as coarsenable as possible (see Section 3.2.5), in order to reap the benefits of multigrid. We enforce a certain degree of "blockiness" in the grids by use of a "blocking factor" F_B, which will be the minimum amount a set of grids can be coarsened. The blocking factor is enforced by coarsening the arrays of cells which are tagged for refinement by F_B before calling the clustering algorithm, which then will produce coarse grids, which are then refined up to the resolution required, as described in Figure 5.4. When coarsening the list of tagged cells, if any fine cell which falls inside a coarsened cell has been tagged for refinement, then the entire coarsened cell is tagged.

5.4.1 Clustering Algorithm

To generate grid configurations from the list of tagged cells, we use the clustering algorithm of Berger and Rigoutsos [20]. In this method, grid generation is an iterative and recursive process. The smallest box possible is placed around the tagged cells. If the grid generated by this box does not satisfy the grid efficiency requirement, then the algorithm looks for a good "cut point" to split the box in two. Cut points are found using an edge detection algorithm that creates a histogram of the number of tagged cells in both the X and Y directions, and then prioritizes cut points by first looking for gaps in tagged cells (where the histogram goes to 0, a natural cut point), and then by looking for places where the second derivative of the histogram changes sign, which is a good indicator of a natural "edge" in the tagged cells. If all else fails, simple bisection of the box is used. Then, cut the initial box along the cut point line, and draw the smallest possible boxes around each of the two subgroups of tagged cells. If either of these boxes does not meet the grid efficiency criterion, then we look for another cut point in the offending box(es). This is continued until we have a set of boxes which all satisfy the grid efficiency criterion.

Chapter 6

Results

This chapter will describe the results of the various test problems we have used to validate the algorithm. The test problems are chosen to demonstrate the convergence properties and robustness of the AMR algorithm. There are various questions which must be answered to demonstrate the effectiveness of the method described in this work. Questions we would like to answer are:

1. Are flow features corrupted when they cross the coarse-fine interface?
2. What is the effect of the volume-discrepancy correction?
3. Do we reach the accuracy of a globally refined calculation through the use of local refinements?

6.1 Test Problem Descriptions

To answer the questions posed in the previous section, we will use three test problems, which are :

1. Steady-state vortex in a box
2. Traveling counter-rotating vortex pair

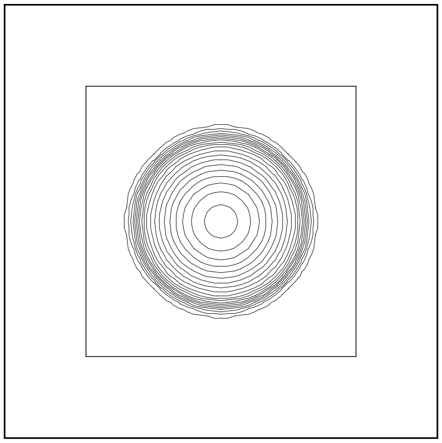


Figure 6.1: Initial vorticity distribution for single vortex problem.

3. Three co-rotating vortices

6.1.1 Single Vortex in a Box

The first test problem is a single steady-state vortex in a box. Initial conditions are given by:

$$u_\theta(r) = \begin{cases} \Gamma(\frac{1}{2}r - 4r^3) & \text{if } r < R \\ \Gamma(\frac{R}{r}(\frac{1}{2}R - 4R^3)) & \text{if } r \geq R \end{cases} \tag{6.1}$$

where r is the radial distance from the vortex center, u_θ is the azimuthal velocity component around the vortex center, R is the radius of the vortex patch, and Γ is the vortex strength. For the single vortex in a box problem, the vortex center is placed at $(x, y) = (\frac{1}{2}, \frac{1}{2})$, $R = 1.0$, and $\Gamma = 0.2$. The initial vorticity distribution for this case is shown in Figure 6.1.

6.1.2 Traveling Vortex Pair

The initial condition is a pair of counter-rotating vortices, each with an initially cubic vorticity profile. The cubic vorticity profile was chosen such that both the vorticity ω and its

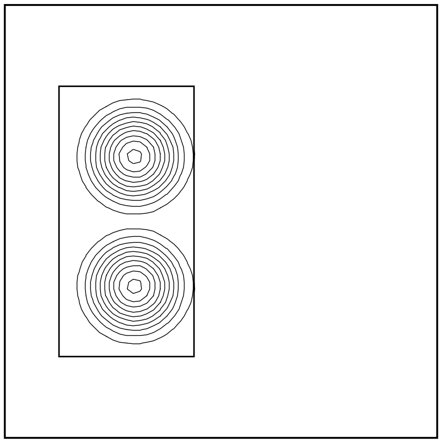


Figure 6.2: Initial vorticity distribution for traveling vortex problem.

derivative $\frac{d\omega}{dr}$ are equal to zero at the radius R from the vortex center, and the circulation of the vortex is equal to $2\pi\Gamma$. The initial velocity for each vortex in this case is given by:

$$u_\theta(r) = \begin{cases} \Gamma\left(\frac{8}{3R^5}r^4 - \frac{5}{R^4}r^3 + \frac{10}{3R^2}r\right) & \text{if } r < R \\ \Gamma(\frac{1}{r}) & \text{if } r \geq R \end{cases} . \tag{6.2}$$

Using superposition, the velocity field induced by each vortex is added together to create the total velocity field. For the test problem in this section, there were two counter-rotating vortices. The first vortex had a strength $\Gamma = 0.35$, a radius $R = 0.15$, and was centered at $(x, y) = (.3, .65)$. The second vortex had a strength of $\Gamma = -0.35$, $r = 0.15$, and was centered at $(x, y) = (.3, .35)$. The initial vorticity distribution is shown in Figure 6.2. Due to the velocity field induced by each vortex, the net effect is that the vortex pair translates to the right.

6.1.3 Three Co-Rotating Vortices

For this test problem, the initial condition is given by three vortices with the cubic vorticity profiles described in Section 6.1.2. In this case, there are three co-rotating vortices. Each vortex had

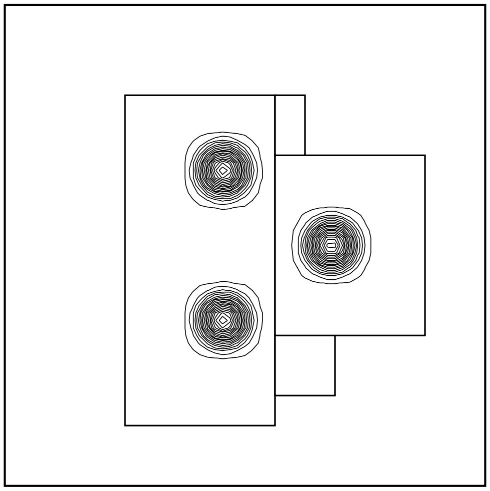


Figure 6.3: Initial vorticity distribution for 3-vortex test case

a strength of $\Gamma = 0.50$ and a radius of $R = 0.75$, and were centered at (0.68,0.5), (0.455, 0.65588457), and (0.455, 0.34411543). The initial vorticity distribution for this case is shown in Figure 6.3. The vortices induce a velocity field which causes the three vortices to revolve around the center of the domain.

6.2 Passage Through Coarse-Fine Interfaces

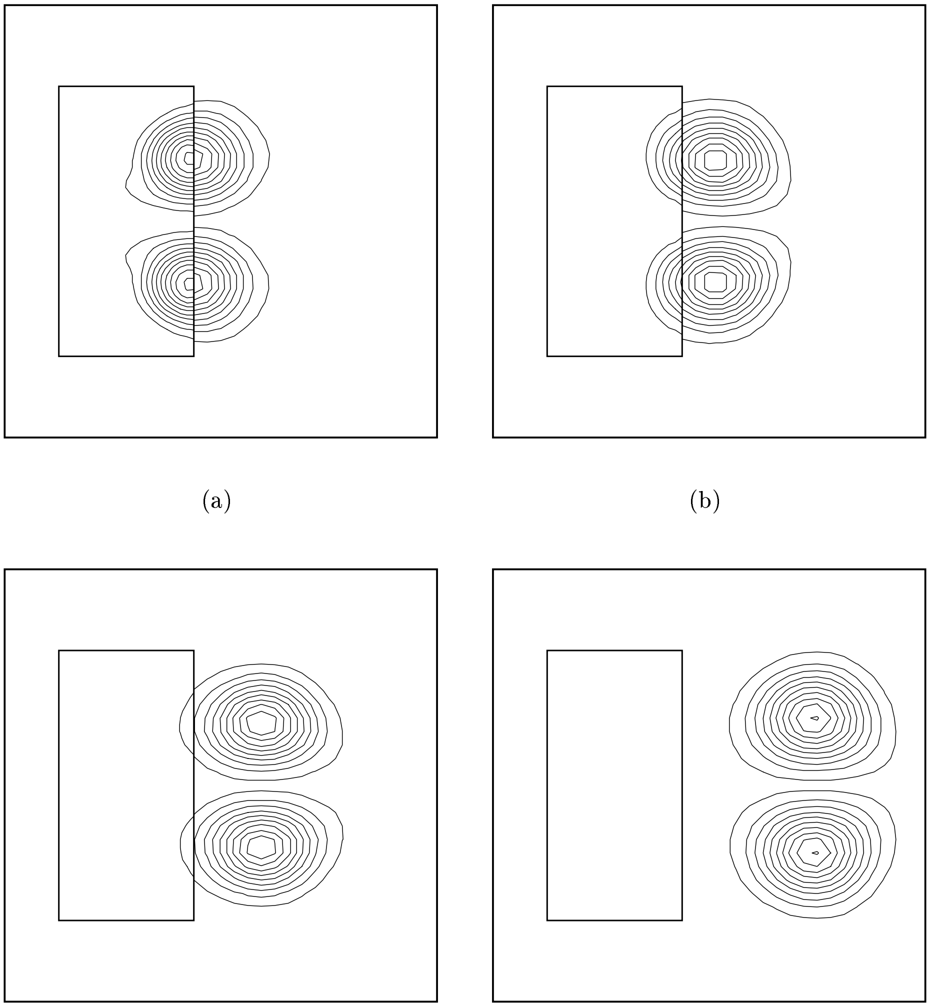
To demonstrate that flow features can pass across coarse-fine interfaces without distortion by the grid discontinuity, we ran the two-vortex test case with a 32×32 base grid and one factor two refinement, but holding the grid configuration constant at the original configuration. In this case, the traveling vortices will translate to the right, crossing the coarse-fine interface and passing onto the coarse grid. This will demonstrate that flow features are not distorted as they cross the coarse-fine interface. Contour plots of the vorticity distribution are shown in Figure 6.4. For comparison purposes, the solution after 150 timesteps of a 32×32 single-grid case is shown in Figure 6.5 As can be seen, there is no noticeable corruption of the vortices as they cross the interface, except for some spreading of the vortices, as is expected due to the coarser resolution. Also, comparing the solutions after 150 coarse timesteps, it is evident that the two solutions do not noticeably differ, so we recover the coarse single-grid solution after passing from the fine patch, which is what we expect.

6.3 Volume-Discrepancy Correction

We would also like to examine the effect of the volume-discrepancy correction described in Section 4.4.2. We will use two test problems. First, we will look at the effects of the volume-discrepancy correction for the steady-state single-vortex problem described in Section 6.1.1. Then, we will examine its effects in a time-dependent case (including the effects of regridding) by looking at the traveling vortex pair problem of Section 6.1.2.

6.3.1 Single Vortex

To isolate the effects of the volume-discrepancy correction without the complications of regridding, we ran the single-vortex test case with and without the volume-discrepancy correction, which corresponded to $\eta = 0.9$ and $\eta = 0$ respectively. A comparison of the distribution of Λ is presented in Figure 6.6. Note that all of the plots in Figure 6.6 have the same scale, which makes the effect of the volume-discrepancy correction evident. Recall that $\Lambda \neq 1$ is a measure of the errors in advection caused by the failure of freestream preservation. It is apparent from Figure 6.6 that without the volume-discrepancy correction, errors in advection are generated at the coarse-fine interface, which are then advected throughout the flow (which in this case is a counter-clockwise rotating vortex), corrupting the solution even away from coarse-fine interfaces. In contrast, with $\eta = 0.9$, the advection errors are confined to the cells immediately adjacent to the coarse-fine



(a) (b)

(c) (d)

Figure 6.4: Vorticity distribution for traveling vortex problem after (a) 50 timesteps, (b) 75 timesteps, (c) 100 timesteps, and (d) 150 timesteps.

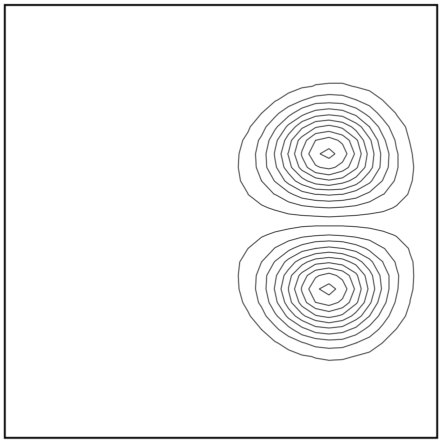
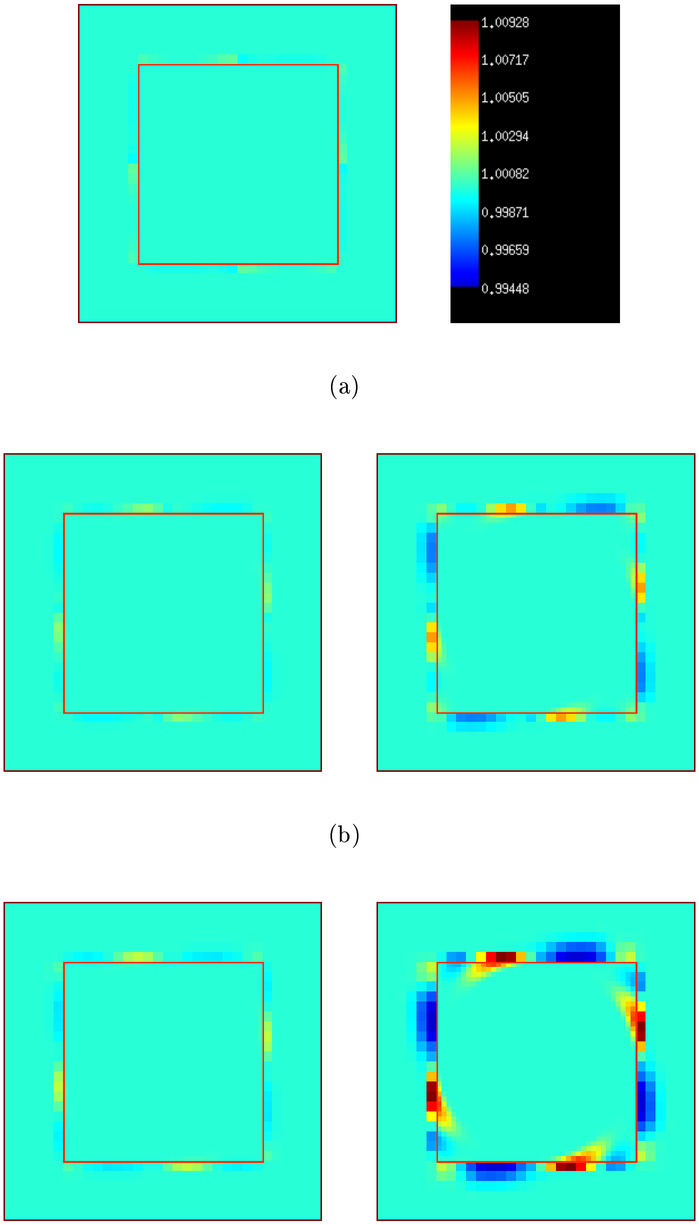


Figure 6.5: 32×32 single-grid case after 150 timesteps

interface, and appear to be kept to the magnitude of the error made in one timestep. Since the volume-discrepancy correction is a lagged one, and as such can only relax errors after they have been made, this is what we would expect.

To further examine the advection errors for the single-vortex case, we ran a series of cases with 32×32, 64×64, and 128×128 base grids, each with one level of refinement. To judge the effects of refinement ratio on the advection errors, each case was run with both $n_{ref} = 2$ and $n_{ref} = 4$. Max(Λ-1), which is the error in Λ, is plotted against time for these cases in Figure 6.7. It is apparent from inspecting Figure 6.7 that, to first approximation, the advection errors are a function of the coarse grid spacing; the effect of the refinement ratio is only secondary. This is especially apparent in the no-correction case. Further inspection of Figure 6.7(a) reveals that without the volume-discrepancy correction, the errors converge at roughly $O(h_c)$, where h_c is the coarse-grid spacing. (The actual order of convergence for this case appears to be between 1.1 and 1.3.) The dips in max(Λ) at $t = 12$ and $t = 25$ arise because the fluid containing the maximum Λ is advected into a source of a deficiency in Λ ($\Lambda < 1$), which causes some cancellation of the extrema of the error.

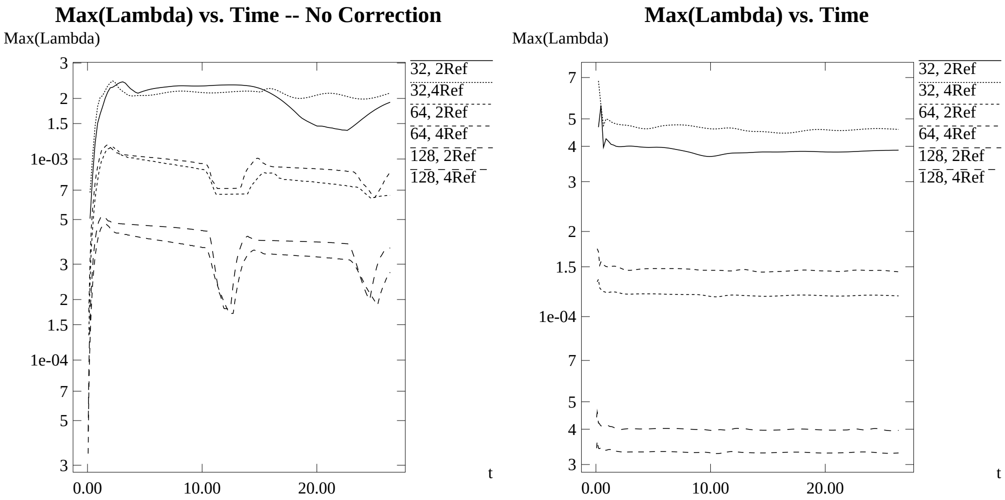


(a)

(b)

(c)

Figure 6.6: Λ after (a) 1 timestep, (b) 10 timesteps, and (c) 20 timesteps. Pictures on left are *with* volume-discrepancy correction, pictures on right are without.



(a) (b)

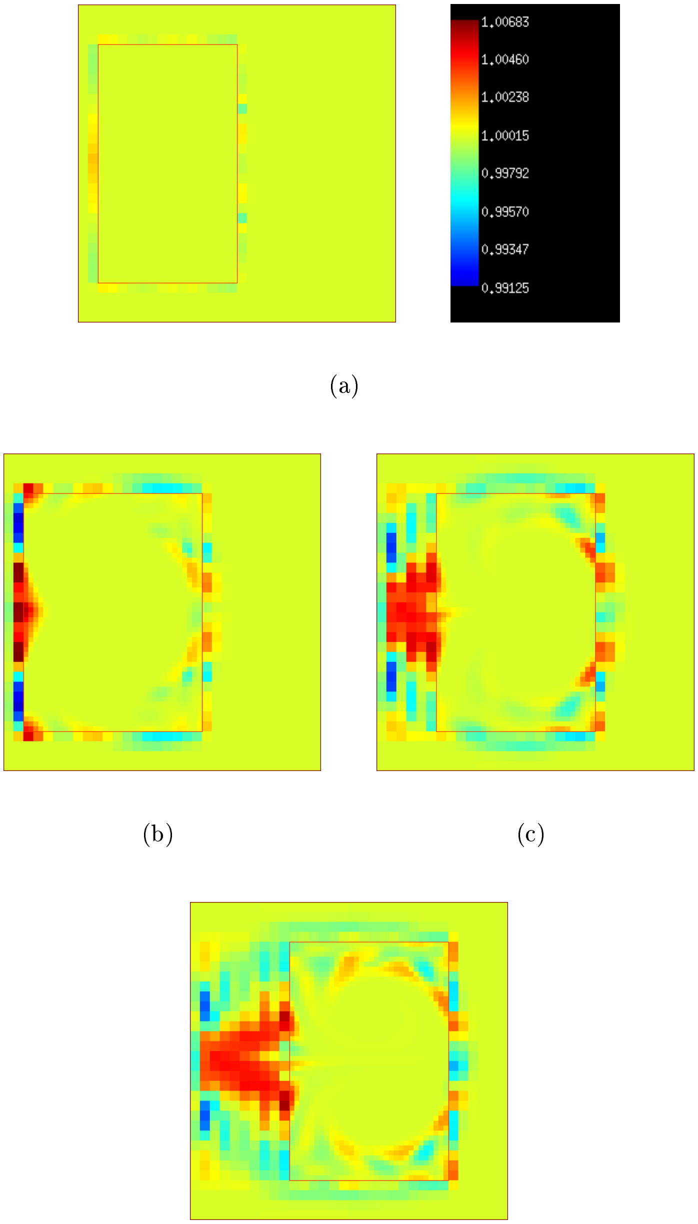
Figure 6.7: Max(Λ) vs. time for the single-vortex case; (a) without volume-discrepancy correction, and (b) with correction. Note that (a) and (b) have different scales.

Comparison of the scales of Figures 6.7(a) and 6.7(b) shows that using the volume-discrepancy correction drastically reduces the maximum error, as was seen in Figure 6.6. As before, it appears that using the correction restricts the error to something less than the error made in one timestep. Since we expect this error to be $O(h_c)\Delta t^c$, where Δt^c is the coarse-grid timestep (which is itself $O(h_c)$), we would expect that this would restore second-order accuracy to this aspect of the method. (The actual convergence rate appears to be around 1.6 between the 32×32 and 64×64 cases, and 1.85 between the 64×64 and 128×128 cases.)

6.3.2 Traveling Vortex Case

Because freestream preservation errors are generated at coarse-fine interfaces, we expect that changing the grid structure as the solution evolves will complicate the issue somewhat. To examine the performance of the volume-discrepancy correction in a fully time-dependent case (including regridding), we repeat the cases in the previous section, but with the initial conditions for the traveling vortex pair of Section 6.1.2. In this case, however, we allow the grids to change dynamically with the solution. For this set of test cases, we will regrid every two coarse-grid timesteps, using the Richardson extrapolation error estimator of Section 5.3.2. For the 32×32 base grid case, we will use an error tolerance of $\tau = 0.8$. Because we expect the truncation error to scale as h^2, we divide this tolerance by $(\frac{h_c}{h_f})^2 = 4$ each time we refine the base grid, so the error estimation tolerance for the 64×64 case will be $\tau = 0.2$, and for the 128×128 case, we will use $\tau = 0.05$. The distribution of the Λ field after 2, 24, 60, and 100 timesteps is shown in Figure 6.8, where no correction was applied, and Figure 6.9, where the correction was applied. Note that the color scales are different for the two figures.

From examining Figure 6.8, it is apparent that as the grids move with the vortices, each



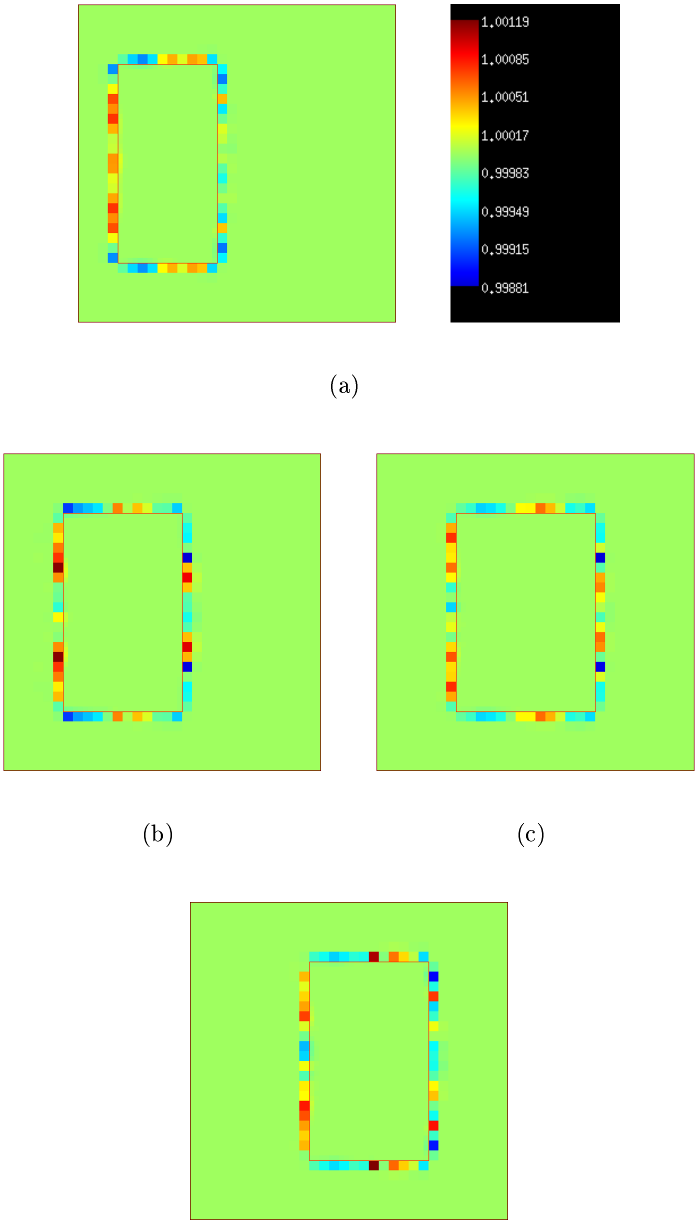
(a)

(b)

(c)

(d)

Figure 6.8: Λ (without volume-discrepancy correction) after (a) 2, (b) 24, (c) 60, and (d) 100 timesteps.



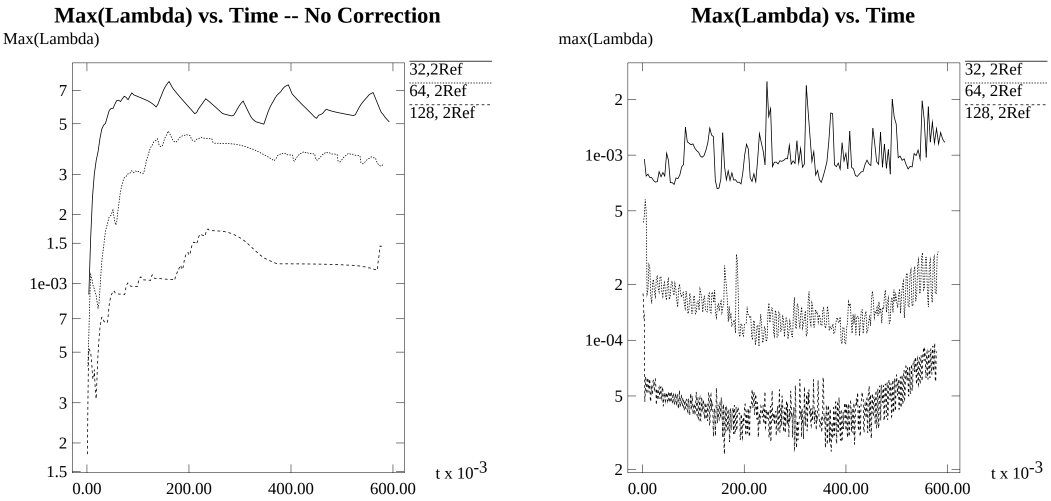
(a)

(b)

(c)

(d)

Figure 6.9: Λ (with volume-discrepancy correction) after (a) 2, (b) 24, (c) 60, and (d) 100 timesteps.



(a) (b)

Figure 6.10: Max(Λ) vs. time for the traveling vortex pair case; (a) without volume-discrepancy correction, and (b) with correction. Once again, note that (a) and (b) have different scales.

.

new coarse-fine interface results in the creation of a new set of errors, which is left behind once the coarse-fine interface has moved. These errors are then advected throughout the flow, as in the single-vortex case. Even with the moving grids, Figure 6.9 shows that the volume-discrepancy correction still confines advection errors to the cells immediately adjacent to the coarse-fine interfaces, and once again, limits them to approximately the error generated in one timestep. While this one-cell-wide error is left behind when the coarse-fine interface moves, it is quickly removed by the action of the volume-discrepancy corrections.

Plots of Max(Λ) *vs.* time are shown in Figure 6.10. Once again, note the radical reduction in the advection error when the volume-discrepancy correction is employed. Note that due to regridding, the max(Λ) plot for the corrected case is much more oscillatory. This is expected because as the grids move, the correction field must adjust to the new grid configuration. In other respects, the results of the time-dependent case behave in the same way as for the steady-state calculation. This points to this technique as a robust method for correcting errors due to the mismatch in advection velocities.

6.4 Accuracy of AMR Calculations

An important question for adaptive methods in general is whether local refinement can result in improved accuracy. In Section 3.2.1 it was demonstrated for Poisson's equation that simply refining the computational mesh without sufficiently linking coarse and fine solutions did not improve the global accuracy of the solution. Likewise, for our algorithm, we would hope that local refinement increases the accuracy of the solution. While we expect some errors due to the reduced accuracy at coarse-fine interfaces, we hope that these errors are outweighed by the gain in accuracy from local refinement. Ideally, local refinement will result in accuracy comparable to that attained in a global fine-grid computation.

To test this, we use the three-vortex problem. Since there is no exact solution for this problem, a 512×512 single-grid computation was performed, which was treated as the "exact" solution for the purposes of this comparison. The errors were computed by averaging the 512×512 solution onto the valid regions of the composite solution, subtracting the composite solution, and then using this composite error to compute the appropriate error norms. In these runs, the global (coarsest-level) timestep was prescribed for each run, so that the solution times would correspond. Five cases were run, each with one level of refinement. Three cases were run with $n_{ref} = 2$, with

Base Grid Size	h= 1/32	1/64	1/128	1/256
2Ref	0.8	0.2	0.05	—
4Ref	0.2	0.05	—	—

Table 6.1: Richardson extrapolation error estimator tolerances used for three-vortex problem

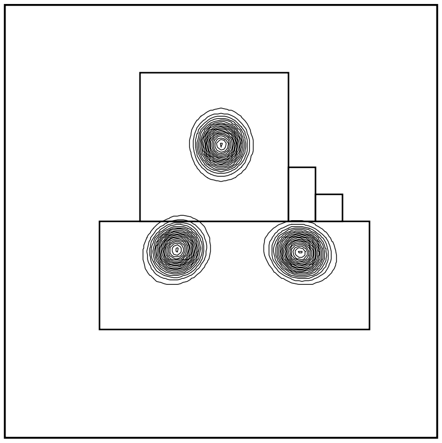


Figure 6.11: Vorticity and grid configuration for three-vortex case, 64×64 base grid, one $n_{ref} = 2$ refinement.

32×32, 64×64, and 128×128 base grids. Also, two cases were run with $n_{ref} = 4$, with 32×32 and 64×64 base grids. To estimate error for grid placement, the Richardson extrapolation error estimator of Section 5.3 was used. Since it was assumed that our method is $O(h^2)$, the tolerance for the error estimator was adjusted to reflect the expected spatial resolution; for example, if the tolerance for a case with spatial resolution $\Delta x = \Delta y = h$ was ϵ, then the tolerance for a case where $\Delta x = \Delta y = \frac{h}{2}$ would be $\frac{\epsilon}{4}$. The error estimation tolerances for each case is shown in Table 6.1

The errors are tabulated in Tables 6.2 and 6.3 for the errors in the $x-$ and $y-$velocities, respectively at $t = 0.128$. The vorticity distribution and grid configurations for the 64×64 base grid, $n_{ref} = 2$ case are shown in Figure 6.11.

Base Grid Size	h= 1/32	1/64	1/128	1/256
Single Grid	0.185132	0.0464363	0.0121967	0.00282103
2Ref	0.0527198	0.0156918	0.00477001	—
4Ref	0.0200913	0.00717611	—	—

(a) L_1(error)

Base Grid Size	h= 1/32	1/64	1/128	1/256
Single Grid	0.490627	0.172484	0.0418936	0.00847492
2Ref	0.171974	0.0423418	0.00978566	—
4Ref	0.0439739	0.011447	—	—

(b) L_2(error)

Base Grid Size	h= 1/32	1/64	1/128	1/256
Single Grid	3.53649	1.55637	0.390388	0.0666353
2Ref	1.51012	0.387115	0.074577	—
4Ref	0.387326	0.0722894	—	—

(c) L_∞(error)

Table 6.2: Errors for x-velocity, time = 0.128

Base Grid Size	h= 1/32	1/64	1/128	1/256
Single Grid	0.178308	0.0461935	0.0119461	0.00277669
2Ref	0.0536107	0.0156112	0.00482953	—
4Ref	0.0195496	0.00746759	—	—

(a) L_1(error)

Base Grid Size	h= 1/32	1/64	1/128	1/256
Single Grid	0.477776	0.171707	0.0421551	0.00849939
2Ref	0.171671	0.0427243	0.00970393	—
4Ref	0.0438788	0.0116263	—	—

(b) L_2(error)

Base Grid Size	h= 1/32	1/64	1/128	1/256
Single Grid	3.47099	1.54005	0.398559	0.0779502
2Ref	1.56308	0.389457	0.0836362	—
4Ref	0.380423	0.0771922	—	—

(c) L_∞(error)

Table 6.3: Errors for y-velocity, time = 0.128

As can be seen, for both times examined, and for both velocity components, local refinement increases the accuracy of the computation as measured in L_1, L_2, and L_∞ norms. For the L_2 and L_∞ norms, the solution error is clearly reduced to a level comparable to the single-grid result with the same resolution. In other words, we see the same error for the 64×64 base grid with one refinement of $n_{ref} = 2$ as for the 128×128 single-grid computation. This is most apparent in the L_∞ norm, where the numbers are almost identical; agreement between AMR results and the equivalent-resolution single-grid results is slightly worse in the L_2 norm, and in the L_1 norm, agreement is often only within a factor of two. This trend is more apparent for the $n_{ref} = 4$ cases. We believe that this is because, while the dominant errors in the solutions are on the interiors of the refined grids around the vortex centers, there are small errors in the solutions which are generated at the coarse-fine interfaces. While these errors do not contribute significantly to the L_∞ and L_2 norms of the error, they accumulate and affect the L_1 norm. One source of error at the coarse-fine error is due to the conservative linear interpolation of the coarse-grid solution used to compute boundary conditions for the advective updates. If the field being interpolated is not well-represented by linear interpolation, some errors will be generated at the coarse-fine interface due to interpolation errors. In the case of velocity advection, these interpolation errors will manifest themselves as filaments of vorticity one fine cell wide which are generated at coarse-fine interfaces and then advected through the flow.

Another source of error in AMR computations, which was mentioned by Almgren et al [5], is coarse-grid errors which are transported into refined regions. This is an inherent problem with locally adaptive methods, since by design, the coarse-grid solution is less accurate than that in refined regions. These errors can be minimized by choosing an appropriate error estimator, which will ensure that the coarse-grid errors are roughly the same scale as errors on the fine grid.

Finally, the creation of new grids through regridding can introduce errors. In many cases, coarse-fine errors at what were coarse-fine interfaces remain behind as a shadow of the previous grid configuration after the grids are moved. These errors are also then advected through the flow. After a series of regridding operations, these errors can be created in many different regions of the domain. Also, in the present code, when a region is newly refined from a coarser level, the coarse solution is simply interpolated to fill the new refined grid. This creates errors on the new refined grid, since the newly interpolated fine solution is not as smooth as if it had been a refined grid already. For instance, after the first post-regridding timestep, the divergence in newly-refined regions will contain a high-frequency component which is eventually damped by repeated application of the projection operators as the solution evolves. When a previously refined region is coarsened, the new valid regions on the coarse grid are filled with the averaged fine solution. In this case, we also see errors due to the fact that the averaging process introduces some error into the coarse solution. In either case, we see an increased error in the newly refined or coarsened regions.

So, while the L_∞ and L_2 norms reflect the improvement of the dominant solution errors around the vortices (which respond well to refinement because they are on the interiors of the refined grids), the lessened responsiveness of the L_1 norm of the error reflects the small errors generated at the coarse-fine interfaces, which are eventually spread through the domain by the background flowfield. This effect is more apparent for the $n_{ref} = 4$ case, because the errors at coarse-fine interfaces are larger, due to the stronger discontinuity in the grid spacing at coarse-fine interfaces for $n_{ref} = 4$.

This is borne out by examination of the error distributions in these cases. If we really *are* achieving fine-grid accuracy, we would expect that the error distributions would look similar. We especially would like to see if solution errors near the coarse-fine interface are noticeably corrupting the solution. For the 64×64 base grid case, Figure 6.12 shows the errors in the x-velocity, while Figure 6.13 shows the errors in the y-velocity at $t = 0.128$. It is apparent that, while the errors look very much like the errors in the corresponding single-grid cases, there is a small but noticeable error which is generated at coarse-fine interfaces and is then spread throughout the flow.

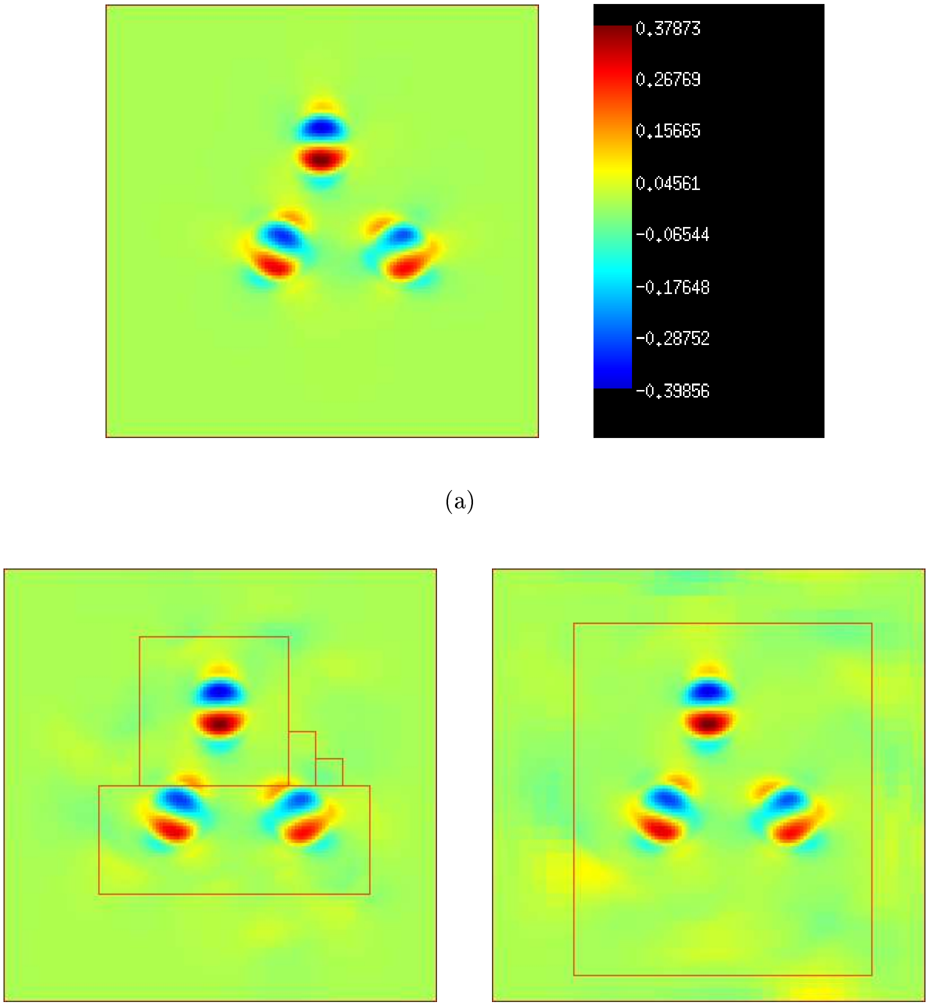
To better show these small AMR errors, Figures 6.14 and 6.15 show the same results, but with a smaller color-scale range, to better show these errors.

It should be noted, however, that while the AMR solutions show some additional error due to coarse-fine interface errors, in all cases, refinement does improve the accuracy of the solution in all norms. In other words, while there is some additional error relative to the uniformly fine-grid solution, the use of AMR does markedly improve the accuracy of the solution relative to the uniformly coarse-grid solution. Also, the errors generated at coarse-fine interfaces are still much smaller than the dominant errors in the solutions, which are still due to solution features, rather than grid boundaries.

Also, some of the coarse-fine interface error could be reduced if the error-estimation criteria took coarse-fine errors into account, as the error-estimation of [67] does; also, it might benefit from the flux-based Richardson extrapolation error-estimation method used by Propp, which takes the surface/volume ratio of refined grids into account.

6.5 Performance

Almgren et al [5] demonstrated that when suitable care is used in optimizing the implementation of an adaptive projection algorithm, that sizeable savings in CPU time could be realized.

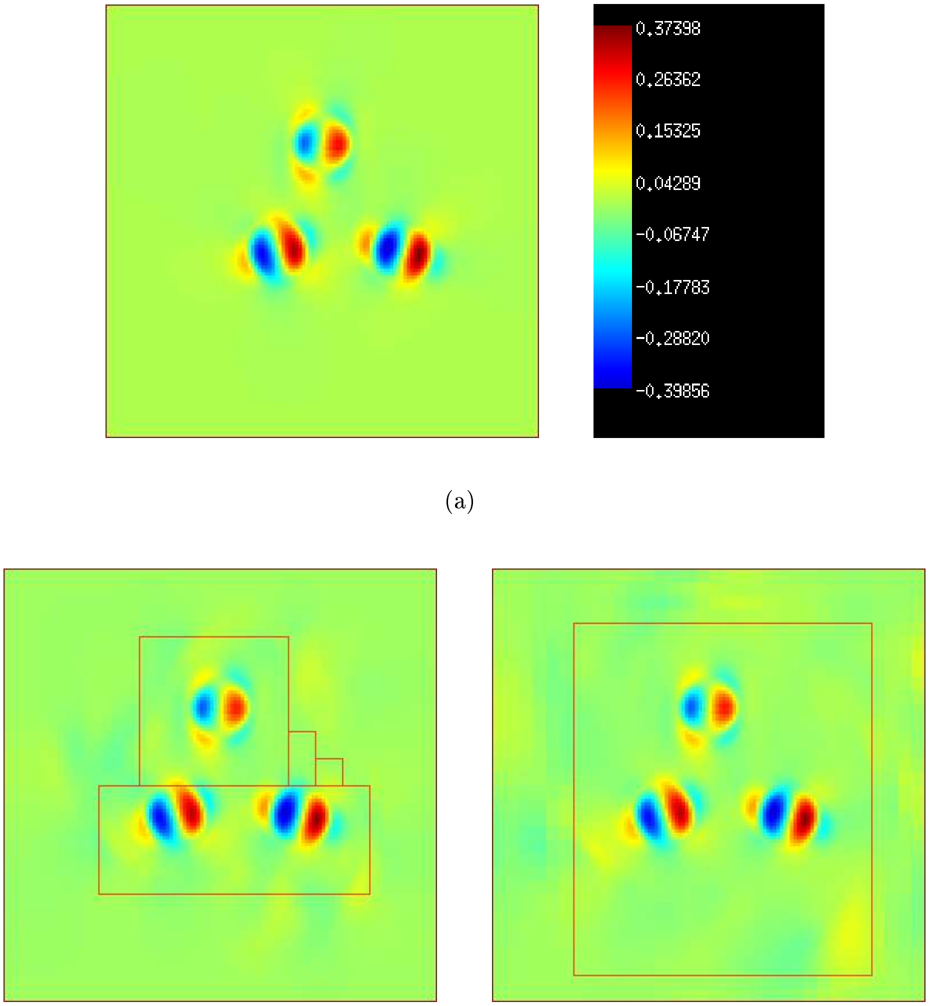


(a)

(b)

(c)

Figure 6.12: Error in x-velocity at t=0.128 for (a) 128×128 single-grid computation, (b) 64×64 base grid with one factor 2 refinement, and (c) 32×32 base grid with one factor 4 refinement.

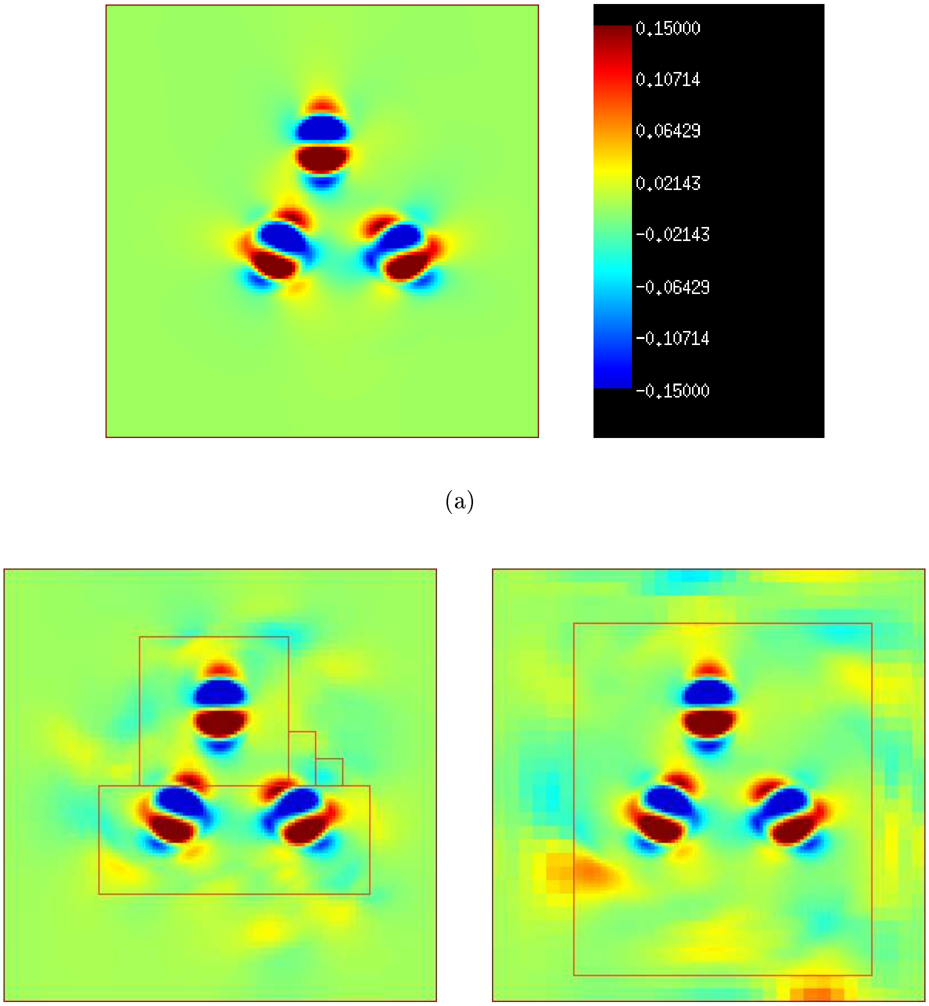


(a)

(b)

(c)

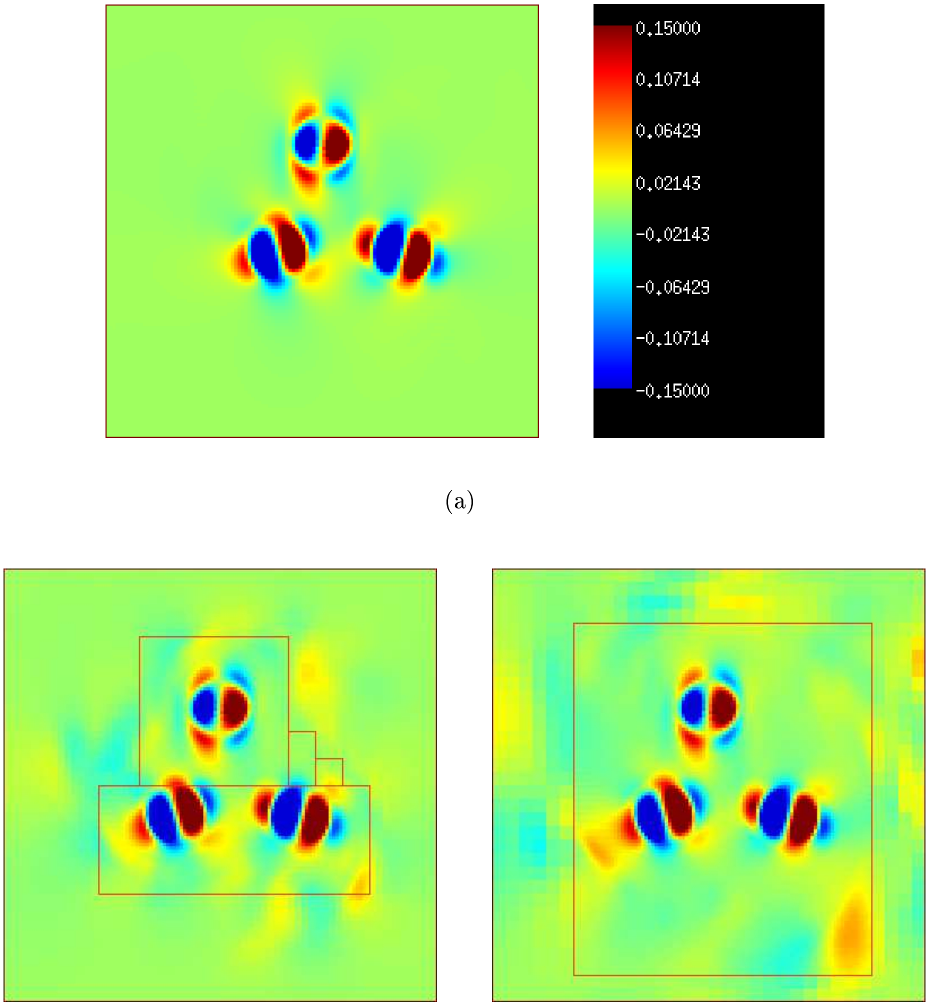
Figure 6.13: Error in y-velocity at t=0.128 for (a) 128×128 single-grid computation, (b) 64×64 base grid with one factor 2 refinement, and (c) 32×32 base grid with one factor 4 refinement.



(a)

(b) (c)

Figure 6.14: Error in x-velocity at t=0.128 for (a) 128×128 single-grid computation, (b) 64×64 base grid with one factor 2 refinement, and (c) 32×32 base grid with one factor 4 refinement. Note that the scale of the color map has been altered to emphasize small errors



(a)

(b)

(c)

Figure 6.15: Error in y-velocity at t=0.128 for (a) 128×128 single-grid computation, (b) 64×64 base grid with one factor 2 refinement, and (c) 32×32 base grid with one factor 4 refinement. Note that the scale of the color map has been altered to emphasize small errors

In light of their results, we would expect that due to the similarity of the algorithms, suitable optimization of the code used in this work would produce similar savings. We do not present timing results here because the purpose of this work was to develop an algorithm which represented a simplification of the algorithm used in [5]. Because the code used in this work was a development code, optimization for CPU efficiency was not performed as part of this work. We expect that after some optimization, the algorithm presented here would present similar savings.

It should be mentioned that the place where the computational costs incurred by the algorithm in this work are much higher than in [5] is in regridding operations. In [5], initializing new fine-level solutions after regridding is performed entirely by interpolation of coarse-level data. In this work, we perform a series of level advances and elliptic solves. Also, the Richardson extrapolation error estimator involves an elliptic solve. So, in most cases, we would tend to regrid less often, and buffer tagged cells more to compensate.

Chapter 7

Software Implementation

Implementing adaptive methods can be difficult, requiring the use of fairly complicated data structures and dynamic memory management to manage computations on the changing grid structure. The use of object-oriented programming techniques, along with the use of a pre-existing software infrastructure made this task manageable.

The algorithms described in this work were implemented in a hybrid of C++ [63] and FORTRAN77 [23]. C++, with its advanced dynamic memory-management and object-oriented capabilities, was used to construct classes which manage the computation. On the other hand, floating-point intensive operations were performed in FORTRAN, to take advantage of the greater optimization of FORTRAN for floating-point operations. The code used in this work, which also represented a sizeable amount of shared infrastructure which was also used by Propp [51] and Bettencourt [22], consisted of 27,300 lines of C++ code and 14,500 lines of FORTRAN code. The header files for this code represented another 8200 lines of code. These numbers do not include the code associated with BoxLib, an infrastructure library used as a base for the code.

7.1 BoxLib

BoxLib [52], a library of C++ classes developed by the Center for Computational Science and Engineering (CCSE) at the Lawrence Berkeley National Laboratory, was an instrumental part of the implementation of this work. BoxLib was developed to facilitate management of computations on unions of logically rectangular grids, and provided an infrastructure which greatly simplified development of this algorithm.

For a single-grid computation, BoxLib provides much of the infrastructure needed to easily implement a projection method. The `IntVect` class provides a convenient way to store and operate on spatial indices. The `Box` class provides functionality for managing logically rectangular regions in space. To store data, the `FArrayBox` class was used. The `FArrayBox` class is a container class for floating point data which provides a convenient way of interfacing FORTRAN and C++. It also contains functionality for operating on floating-point data directly.

While it provides a convenient infrastructure for implementing single-grid algorithms, for managing adaptive computations on a dynamically changing hierarchy of grids, BoxLib proved indispensable. The `BoxArray` class, which is an array of `Box`'s, proved useful for describing the union of rectangles which make up a refined level. Additionally, the `MultiFab` class, an array of `FArrayBox`s with many additional features, was quite useful for organizing and operating on data on the unions of logically rectangular grids which make up a level.

7.2 Managing the AMR hierarchy

In many ways, the basic structure of this code borrowed heavily from the adaptive implementation of Almgren et al. [5]. In general, our strategy has been to use a single parent AMR class

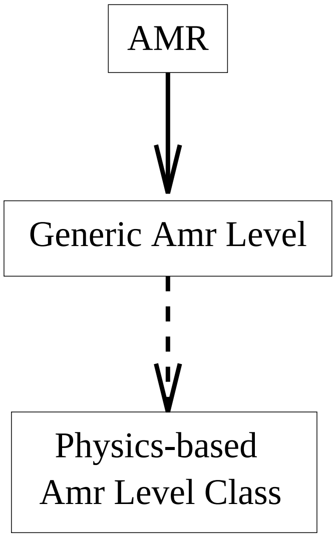


Figure 7.1: Basic class structure for AMR computations. Solid arrows indicate membership, while dashed lines indicate derivation. In this figure, the AMR class contains a generic AMR Level class (actually, an array of them), and the physics-dependent class is derived from the generic Amr Level class.

to manage the hierarchy of levels. This parent class contains an array of AMR level classes, which manage the solution on individual levels of refinement. Since much of the functionality needed by level classes is generically applicable to a broad class of adaptive algorithms, while other functionality and implementation details are specific to a given problem being solved, broad use was made of the inheritance features of C++ by defining a generic level-based class which contained the basic functionality for managing a solution on an AMR refinement level and then deriving problem-specific physics-based classes from the general AMR level class. See Figure 7.1. For more on the use of derivation in the design of AMR classes, see Crutchfield and Welcome [32].

We also extensively use the concept of level-operator classes. A level-operator will contain all the functionality and auxiliary data structures necessary to apply an operator on a level. For the Euler equations, examples of level operator classes would be classes which manage the advection and projection operations done during the advance on a level.

7.2.1 Elliptic Solver

Design of the elliptic solvers was complicated by the desire to have a solver which could be used both as a stand-alone elliptic solver (used for Poisson's equation in [47] and for the steady-state drift-diffusion equations in [22]) and as a solver which could be included in a time-dependent AMR code (which was used in this work, and for the solution of porous media flow in a trickle-bed reactor in [51]). This dual objective was realized through the use of derivation.

First, a set of classes was developed to provide elliptic solver capability for a generic AMR system. The class `AMRSolver` manages the hierarchy of levels and provided an interface to the elliptic solver. The `AMRSolver` class contains an array of `LevelSolver` classes, which manage the solution on each AMR level. In the time dependent case, the `AMRSolver` is a statically defined object which exists parallel to the time-dependent AMR hierarchy. The `AMRSolver` class contains all interface functionality necessary to perform single-level or multi-level solves with $\ell_{base} \geq 0$ (see Chapter 3), and to modify the solver's grid hierarchy as it changes during the time-dependent solution evolution.

For a stand-alone elliptic solver, some capability had to be added to extend the elliptic solver classes for use in this context. For example, while the `LevelSolver` classes manage the solution on individual levels, they do not actually "own" the memory for the solution or right-hand-side, for efficiency reasons. So, to construct a stand-alone solver, an `AMRPoisson` class was derived from the `AMRSolver` class, and a `LevelPoisson` class was derived from the `LevelSolver` class (see Figure 7.2). This greatly simplified code-development and re-use, because the same base classes were used for both the time-dependent and stand-alone solvers.

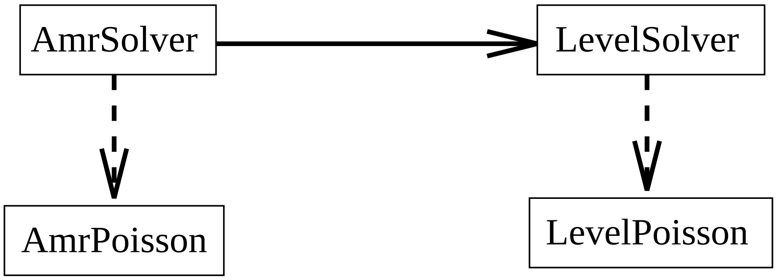


Figure 7.2: Class structure for elliptic solver classes. Solid arrows indicate membership, while dashed arrows indicate derivation.

7.2.2 Euler Equation Code

As was mentioned earlier, a great deal of the functionality necessary for managing time-dependent AMR computations is common across specific physical problem implementations. For example, the functionality used to regrid, or to subcycle in time, is common to all implementations which follow the basic AMR model that we use. On the other hand, many operations, such as advancing on a level, are specific to the set of equations being solved. For this reason, the derivation and virtual function features of C++ were exploited heavily in the implementation of the algorithms in this work. The `Amr` and `AmrLevel` classes used as a starting point for the implementation of this work were modified versions of classes originally developed by CCSE for solving time-dependent problems using AMR [5, 50]. A schematic of the class structure used in the implementation of this algorithm is shown in Figure 7.3.

The `Amr` class, which is very similar to the class used by [5], manages the entire AMR hierarchy and the global time-stepping. For the Euler equations, the `Amr` class also allocates the statically defined `AmrSolver` class, which will manage all elliptic solves. The `Amr` class contains an array of `AmrLevel` classes, which contain the basic functionality to manage the solution on an AMR

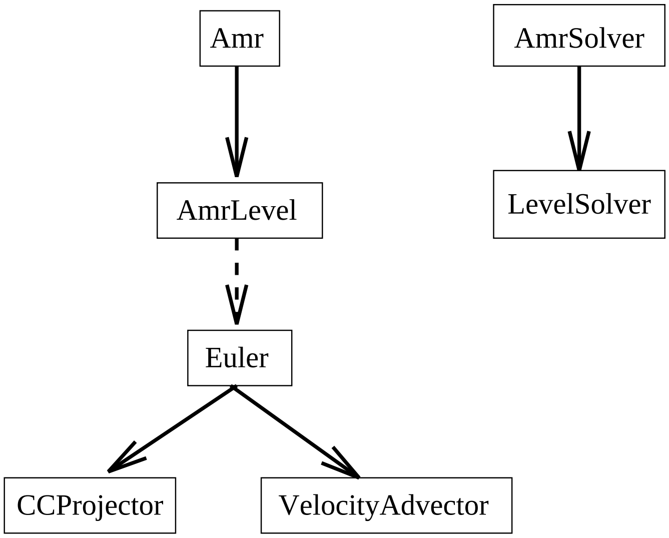


Figure 7.3: Class structure for time-dependent incompressible Euler classes. Solid arrows indicate membership, while dashed arrows indicate derivation.

level. From the parent `AmrLevel` class is derived an `Euler` class, which contains the functionality necessary to implement the specific algorithm being implemented, in this case, the cell-centered projection method described in Chapter 4. The `Euler` class allocates several level-operator classes as necessary, including the `CCProjector` and `VelocityAdvector` classes, which manage the projection and advection parts of the algorithm, respectively. This design allows for greater modularity of the different components of the algorithm, and made it easier to share code with other developers (for example, the code used to implement the algorithm described in this work, that of Bettencourt [22], and that of Propp [51] shared much of the basic infrastructure).

7.3 Visualization

For code-development and examination of the results of AMR computations, good data visualization is indispensable. We used two different visualization tools. AmrVis [14] was developed by CCSE for the purpose of presenting results of AMR computations, and proved to be quite useful for examining the results of finished computations. All color pictures of AMR results in this work were generated using AmrVis.

For examining data during runs (for example, while debugging the code), the VIGL graphics library [33], as extended by Hans Johansen for use with BoxLib, also proved quite useful.

Chapter 8

Conclusions

8.1 Summary

This thesis presents an adaptive cell-centered projection method for the incompressible Euler equations in two dimensions. We use block-structured local refinement in space and time to reduce the amount of computational resources needed to compute numerical solutions with adequate resolution.

First, a single-grid algorithm was presented, which uses the projection formulation of Bell, Colella, and Glaz [16] to construct a second-order projection method which uses the approximate projection discretization of Lai. Results were presented which indicated that the method is second-order accurate.

Then, a multilevel algorithm for solving Poisson's equation on a multilevel hierarchy of locally refined grids using multigrid-accelerated point relaxation was presented. We introduced the notions of composite operators, which operate on variables defined on the multilevel hierarchy of grids, and level operators, which operate on variables defined on single refined levels. It was described how the use of composite operators will enable solutions to Poisson's equation to attain the increased accuracy expected from a locally-refined solution. Results of a test computation were presented to demonstrate the benefits of local refinement on CPU time and memory usage. Also, an alternate algorithm was presented which demonstrated the benefits of multilevel solution approach for elliptic equations, rather than solving the equations using level operators and then computing corrections to ensure that the solution satisfies the equations based on composite operators.

The single-grid projection algorithm was then extended to the solution of the incompressible Euler equations on a multilevel hierarchy of grids. Like the algorithm of Almgren et al. [5], the solution on finer grids is advanced at a finer timestep than that on coarser grids, and then is synchronized with coarser levels when coarse and fine solutions reach the same time. The algorithm described in this thesis differs from that of Almgren et al. in three major respects. For the projection method described in this work, the synchronization step involves a synchronization projection based on cell-centered composite operators to ensure that the composite solution satisfies the divergence constraint based on composite operators. Also, to correct for errors in advection due to the presence of coarse-fine interfaces, we employ a volume-discrepancy correction based on the scheme presented by Propp [51] for porous media flows to compute a lagged correction which approximately corrects for errors in advection. Finally, all elliptic solves performed during synchronization operations are constructed as multilevel solves over all levels which have reached the same time. Because all elliptic solves are based on cell-centered discretizations, it is expected that extension of this algorithm to more complicated problems should be simplified.

Then, strategies for identifying regions which could benefit from local refinement are presented. Refined grids can be placed in user-defined locations, or refinements can be adaptively placed using user-defined criteria (usually based on solution features) or an estimate of local truncation error based on Richardson extrapolation. Computation of estimates of the local truncation error for Poisson's equation involves comparing the Laplacian operator applied to the solution with a coarsened Laplacian operator applied to a coarsened solution. For time-dependent problems, we compute coarse and fine approximations to the discrete update equation, and then compare them. It was noted that using Richardson extrapolation to estimate error, rather than using solution-based estimators, allows the sources of solution error to be localized, which should improve the effectiveness of local refinement.

The adaptive algorithm described in this work was applied to a series of test problems to demonstrate its effectiveness. It was shown that solution features are not corrupted as the cross coarse-fine interfaces. Also, it was shown that the $O(h)$ advection errors due to coarse-fine mismatches in advection velocities can be reduced to $O(h^2)$ by using the volume-discrepancy correction scheme used in this work. Finally, it was demonstrated that using local refinements can enable solution accuracy comparable to the equivalent uniform fine-grid solutions. Solution errors due to local refinement are presented as well. These errors, while enough to prevent the L_1 norm of the errors in locally refined solutions from completely matching single-grid errors, are still uniformly small, compared to other feature-based solution errors.

8.2 Conclusions and Future Work

We have shown that the method presented in this work is effective at modeling the simple test cases presented in this work. In particular, we have demonstrated that the error caused by the addition of adaptivity is small in relation to other solution errors for the problems examined, and that the volume-discrepancy correction is useful in reducing advection errors to approximately the errors resulting from one uncorrected timestep.

The adaptive algorithm presented in this work is intended as the first step in a series of extensions. The first, most obvious step is the extension to the incompressible Navier-Stokes equations by adding viscosity. In a similar vein, diffusion processes should be added to the advected scalar evolution equations. Also, extending this work to solve the equations of variable-density incompressible flow would allow many more physical situations to be modeled.

Extension of this work to more realistic geometries would be useful. Addition of embedded-boundary Cartesian grid techniques, like the ones employed in [50], would allow modeling of flows in more complicated geometries. It is expected that extension of this algorithm to the embedded boundary case will be simplified by the fact that there is only one set of (cell-centered) solvers which need be extended. Also, along the lines of more realistic geometries would be the extension of this algorithm to three-dimensions, which should be straightforward.

The coarse-fine errors seen in the computations indicate the need for better error-estimation techniques, ones which will account for the presence of errors due to coarse-fine interfaces. For example, the flux-based Richardson extrapolation error estimation technique presented in Propp [51], which includes the surface-to-volume ratio of grids in its error-estimation scheme, could prove useful.

Finally, there is the issue of regridding. In this work, a solution is created on new fine-level grids by simply interpolating the existing coarse-level solution to the fine-level resolution. There is evidence that the fine-level solution produced in this way is not smooth enough, and that a better way of initializing new grids with a smooth solution is needed. It is expected that this will become more important when solving viscous flows, since while the interpolated solution is somewhat smooth, the second derivatives of the interpolated solutions is not.

Bibliography

[1] G. Acs, S. Doleschall, and E. Farkas. General purpose compositional model. *Society of Petroleum Engineers Journal*, pages 543–552, August 1985.

[2] M. J. Aftosmis. Upwind method for simulation of viscous flow on adaptively refined meshes. *AIAA Journal*, 32(2):268–77, Feb 1994.

[3] M.J. Aftosmis, M.J. Berger, and J.E. Melton. Robust and efficient Cartesian mesh generation for component-based geometry. *AIAA Journal*, 36(6):952–960, June 1998.

[4] A. Almgren, J. Bell, and W. Szymczak. A numerical method for the incompressible Navier-Stokes equations based on an approximate projection. *SIAM Journal of Scientific Computing*, 17:358–369, 1996.

[5] A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *Journal of Computational Physics*, 142:1–46, 1998.

[6] Ann Almgren. private communication, 1998.

[7] Ann Almgren, Thomas Buttke, and Phillip Colella. A fast adaptive vortex method in three dimensions. *Journal of Computational Physics*, 113:177–200, 1994.

[8] A.S. Almgren, J.B. Bell, P. Colella, and T. Marthaler. A Cartesian grid projection method for the incompressible Euler equations. *SIAM Journal of Scientific Computing*, 18(5):1289=309, Sept. 1997.

[9] David C. Arney and Joseph E. Flaherty. An adaptive level mesh refinement method for time-dependent partial differential equations. *Applied Numerical Mathematics*, 5:257–274, 1989.

[10] D. Bai and A. Brandt. Local mesh refinement multilevel techniques. *SIAM J. Sci. Stat. Comput.*, 8(2):109–134, 1987.

[11] Timothy J. Baker. Mesh adaptation strategies for problems in fluid dynamics. *Finite Elements in Analysis and Design*, 25:243–273, 1997.

[12] R. Barret, M.Berry, T.F.Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia, PA, 1994.

[13] G.K. Batchelor. *An Introduction to Fluid Dynamics.* Cambridge University Press, 1967.

[14] V. Beckner. *Amrvis User's Guide.* Center for Computational Science and Engineering, National Energy Research Supercomputing Center, Lawrence Berkeley National Laboratory, 1996. Available at http://seesar.lbl.gov/ccse/software/software.html.

[15] J. B. Bell, M. J. Berger, J. Saltzman, and M. L. Welcome. Three dimensional adaptive mesh

refinement for hyperbolic conservation laws. *SIAM Journal of Scientific Computing*, 15(1):127–138, 1994.

[16] J. B. Bell, P. Colella, and H. M. Glaz. An efficient second-order projection method for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 85:257–283, 1989.

[17] J.B. Bell, P. Colella, and L. H. Howell. An efficient second-order projection method for viscous incompressible flow. In *Proceedings, AIAA 10th Computational Fluid Dynamics*, pages 360–367, Honolulu, HI, June 1991.

[18] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 1989.

[19] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[20] M. J. Berger and I. Rigoutsos. An algorithm for point clustering and grid generation. *IEEE Transactions Systems, Man and Cybernetics*, 21(5):1278–1286, 1991.

[21] M.J. Berger and A. Jameson. Automatic adaptive grid refinement for the Euler equations. *AIAA Journal*, 23:561–568, 1985.

[22] M. T. Bettencourt. *A Block Structured Adaptive Steady-State Solver for the Drift Diffusion Equations.* PhD thesis, U.C. Berkeley, 1998.

[23] G. J. Borse. *Fortran 77 and Numerical Methods for Engineers.* PWS Publishers, 1985.

[24] Achi Brandt. Multilevel adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[25] W. L. Briggs. *A Multigrid Tutorial.* SIAM, 1987.

[26] D.L. Brown and M.L. Minion. Performance of under-resolved two-dimensional incompressible flow simulations. *Journal of Computational Physics*, 122(1):165–83, 1995.

[27] Carolyn Jean Chee. *A Numerical Model of Blood Flow Through the Mitral Valve.* PhD thesis, U.C. Berkeley, 1998.

[28] G. Chesshire and W. D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *Journal of Computational Physics*, 90(1):1–64, September 1990.

[29] A.J. Chorin. Numerical solutions of the Navier-Stokes equations. *Mathematics of Computation*, 22:745–762, 1968.

[30] Terry L. Clark and R.D.Farley. Severe downslope windstorm calculations in two and three spatial dimensions using anelastic interactive grid nesting: A possible mechanism for gustiness. *Journal of Atmospheric Sciences*, 41(3):329–350, 1984.

[31] R. Courant, K. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM Journal*, 11:215–234, 1967. English translation of the original work, "Uber die Partiellen Diefferenzengleichungen der Mathematischen Physik," Math. Ann. 100, 32-74 (1928).

[32] W. Y. Crutchfield and M. L. Welcome. Object oriented implementation of adaptive mesh refinement algorithm. *Scientific Programming*, 2(4), 1993.

[33] A. B. Downey. VIGL 3.0: Visualization graphics library. Technical report, Computer Science Division, University of California, Berkeley, August 1995. Code and documentation available at `http://http.cs.berkeley.edu/ downey/vigl/`.

[34] Scott Dudek. *A Structured-Grid Adaptive Mesh Refinement Multigrid Algorithm for Steady-State Flows.* PhD thesis, U.C. Berkeley, 1996.

[35] W.N. E and J.G. Liu. Full numerical simulation of coflowing, axisymmetric jet diffusion flames. *SIAM J. Numer. Anal.*, 2(5):720–728, 1995.

[36] C.A.J. Fletcher. *Computational Techniques for Fluid Dynamics*, volume II. Springer-Verlag, 1991.

[37] P.M. Gresho and R.L. Sani. On pressure boundary conditions for the incompressible Navier-Stokes equations. *Int. J. for Numer. Methods Fluids*, 7:1111–1145, 1987.

[38] F.H. Harlow and J.E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluids with free surfaces. *Physics of Fluids*, 8(12), 1965.

[39] James Hilditch. *A Projection Method for Low Mach Number Reacting Flow in the Fast Chemistry Limit.* PhD thesis, U.C. Berkeley, 1997.

[40] R. D. Hornung and J. A. Trangenstein. Adaptive mesh refinement and multilevel iteration for flow in porous media. *Journal of Computational Physics*, 136:522–545, 1997.

[41] L.H. Howell and J. B. Bell. An adaptive mesh projection method for viscous incompressible flow. *SIAM Journal of Scientific Computing*, 18(4):996–1013, 1997.

[42] Jr J.E. Dendy. Black box multigrid for systems. *Appl. Math. Comp.*, pages 67–74, 1986.

[43] H. S. Johansen. *Cartesian Grid Embedded Boundary Finite Difference Methods for Elliptic and Parabolic Equations on Irregular Domains.* PhD thesis, U.C. Berkeley, 1997.

[44] Mindy Lai. *An Approximate Projection Method for Reacting Flow in the Zero Mach Number Limit.* PhD thesis, U.C. Berkeley, 1994.

[45] D. Lee and Y.M. Tsuei. A hybrid adaptive gridding procedure for rescirculating fluid flow problems. *Journal of Computational Physics*, 108:122–141, 1993.

[46] R. J. LeVeque. *Numerical Methods for Conservation Laws.* Birkhäuser, 1990.

[47] D. F. Martin and K. L. Cartwright. Solving Poisson's equation using adaptive mesh refinement. Technical Report UCB/ERL M96/66, UC Berkeley, 1996.

[48] Michael L. Minion. A projection method for locally refined grids. *Journal of Computational Physics*, 127(1):158–178, Aug. 1996.

[49] M.L. Minion and D.L. Brown. Performance of under-resolved two-dimensional incompressible flow simulations. II. *Journal of Computational Physics*, 138(2):734–65, 1997.

[50] R. B. Pember, J. B. Bell, P. Colella, W. Y. Crutchfield, and M. L. Welcome. Cartesian grid method for unsteady compressible flow in irregular regions. *Journal of Computational Physics*, 120:278–304, 1995.

[51] Richard Propp. *Numerical Modeling of a Trickle Bed Reactor.* PhD thesis, UC Berkeley, 1998.

[52] C. Rendleman, V. Beckner, J. B. Bell, W. Y. Crutchfield, L. Howell, and M. L. Welcome. *BoxLib User's Guide and Manual: A Library for Managing Rectangular Domains.* Center for Computational Science and Engineering, National Energy Research Supercomputing Center, Lawrence Berkeley National Laboratory, 1996. Available at http://seesar.lbl.gov/ccse/software/software.html.

[53] W. J. Rider. Filtering nonsolenoidal modes in numerical solutions of incompressible flows. Technical Report LA-UR-3014, Los Alamos National Laboratory, 1994.

[54] W.J. Rider. Approximate projection methods for incompressible flow: Implementation, variants, and robustness. Technical Report LA-UR-2000, Los Alamos National Laboratory, 1994.

[55] W.J. Rider. The robust formulation of approximate projection methods for incompressible flows. Technical Report LA-UR-3015, Los Alamos National Laboratory, 1994.

[56] H. Schlichting. *Boundary Layer Theory.* McGraw-Hill Bool Company, Inc., 1960.

[57] William C. Skamarock and Joseph B. Klemp. Adaptive grid refinement for two-dimensional and three-dimensional nonhydrostatic atmospheric flow. *Monthly Weather Review*, 122:788–804, 1993.

[58] Erlendur Steinthorsen, David Modiano, William Y. Crutchfield, John B. Bell, and Phillip Colella. An adaptive semi-implicit scheme for simulations of unsteady viscous compressible flow. In *AIAA Paper 95-1727-CP, Proceedings of the 12th AIAA CFD Conference*, 1995.

[59] David E. Stevens. *An Adaptive Multilevel Method for Boundary Layer Meteorology.* PhD thesis, Dept. of Applied Mathematics, University of Washington, 1994.

[60] David E. Stevens and Christopher S. Bretherton. A forward-in-time advection scheme and adaptive multilevel flow solver for nearly incompressible atmospheric flow. *Journal of Computational Physics*, 129:284–295, 1996.

[61] J.C. Strikwerda. Finite difference methods for the Stokes and Navier-Stokes equations. *SIAM J. Sci. Stat. Comput.*, 5:56–67, 1984.

[62] John C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations.* Wadsworth & Brooks/Cole Advanced Books and Software, 1989.

[63] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley Publishing Company, 1991.

[64] P.K. Sweby and H.C. Yee. On the dynamics of some grid adaption schemes. In *Proc. 4th Int. Conf. on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, 1994.

[65] M.C. Thompson and J.H. Ferziger. An adaptive multigrid technique for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 82:94–121, 1989.

[66] J. A. Trangenstein and J. B. Bell. Mathematical structure of the black-oil model for petroleum reservoir simulation. *SIAM Journal of Applied Math*, 49(3):749–783, 1989.

[67] R.A. Trompert and J.G. Verwer. Analysis of the implicit Euler local uniform grid refinement method. *SIAM Journal of Scientific Computing*, 14:259–278, 1993.

[68] Bram van Leer. Towards the ultimate conservative differences scheme IV: a new approach to numerical convection. *Journal of Computational Physics*, 23:263–275, 1977.

[69] P. Wesseling. *An Introduction to Multigrid Methods.* John Wiley and Sons, 1991.